

AFRL-IF-RS-TR-2003-37
Final Technical Report
February 2003



APPLICATION OF INTRUSION TOLERANCE TECHNOLOGY TO JOINT BATTLESPACE INFOSPHERE (JBI)

University of Colorado

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

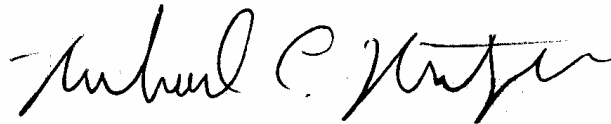
This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2003-37 has been reviewed and is approved for publication.

APPROVED: 

WALT TIRENIN
Project Engineer

FOR THE DIRECTOR:



RICHARD C. METZGER
JBI Program Manager
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> <i>OMB No. 074-0188</i>	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE FEBRUARY 2003	3. REPORT TYPE AND DATES COVERED Final Dec 00 – Jun 02	
4. TITLE AND SUBTITLE APPLICATION OF INTRUSION TOLERANCE TECHNOLOGY TO JOINT BATTLESPACE INFOSPHERE (JBI)			5. FUNDING NUMBERS C - F30602-01-1-0503 PE - 62702F PR - OIPS TA - BA WU - P1	
6. AUTHOR(S) Alexander Wolf, John Knight, Antonio Carzaniga, and Dennis Heimbigner				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) The Regents of the University of Colorado Office of Contracts and Grants 3100 Marine Street, RM 481 Boulder Colorado 80309			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory/IFSE 525 Brooks Road Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2003-37	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Wladimir Tirenin/IFSE/(315) 330-1871/ Wladimir.Tirenin@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) Military information systems, such as a JBI, have evolved to a point where military organizations rely heavily upon them. In fact, the ability of the Department of Defense to use its resources effectively is contingent on the proper operation of these information systems. Improving the survivability of critical military information systems is essential for military applications. The Willow survivability architecture is designed to improve information system survivability significantly. The way in which this is done is to implement a monitoring and control structure that operates separately from the information system itself. The survivability mechanism is responsible for detecting faults and recovering from them. The Willow architecture is a general concept and the goal of this project was to determine its utility and relevance to the JBI concept. The overall conclusion is that all aspects of Willow technology are useful to the JBI.				
14. SUBJECT TERMS Joint Battlespace Infosphere, Information System Survivability			15. NUMBER OF PAGES 129	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

CONTENTS

1. Introduction	1
2. Project Description	2
2.1. Introduction	2
2.2. Critical distributed applications and survivability	3
2.3. Types of fault	4
2.4. Willow concepts	5
2.5. The Willow architecture	7
2.6. Network Simulation Software	12
2.7. Overall Evaluation	15
2.8. Related Work.	18
2.9. References for This Section	19
3. Summary of Research Findings.	21
3.1. Willow Architecture And JBI Survivability	21
3.2. Probabilistic Risk Analysis And JBI Security	21
3.3. Publish/subscribe Security	22
3.4. Non-Military JBI Applicability.	22
4. Personnel Associated with Research Effort.	23
5. Publications	25
6. Presentations Given	27
7. Discoveries, Inventions, and Patents	28
Appendix A Domain Analysis of the Joint Battlespace Infosphere	29
Appendix B The Application of Probabilistic Risk Analysis to Security.	51
Appendix C Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems	111

LIST OF FIGURES

Figure 1. Willow survivability architecture concept.	6
Figure 2. The Willow architecture showing major components and execution-time event communications	7
Figure 3. Organization of the Trust Broker mechanism	12
Figure 4. The RAPTOR modeling system architecture	13
Figure 5. A Model of Multiple Interacting Infrastructure Systems	14
Figure 6. RAPTOR Virtual Message Processor	15
Figure 7. System 2 JBI Implementation	18

1. INTRODUCTION

This is the final report for Air Force Rome Laboratories grant number F30602-01-1-0503, “Application of Intrusion Tolerance Technology to JBI”. The period of the grant was December 12, 2000 to June 11, 2002. The principal investigator on this grant was Dr. Alexander Wolf of the Department of Computer Science, University of Colorado, Boulder, Colorado 90309-0430. Other principal team members were Antonio Carzaniga (University of Colorado), Dennis Heim-bigner (University of Colorado), and John Knight (University of Virginia).

The goal of this research was to determine the applicability of the research being conducted on the Willow survivability architecture to the Joint Battlespace Infosphere (JBI) concept. The approach taken was to examine the various technologies of the Willow system to determine how they might be applied, and to evaluate the technologies by developing two prototype JBI systems.

Two activities were undertaken that were somewhat outside the scope of the basic evaluation of the Willow architecture. The first was to examine the JBI concept so as to understand it. This was undertaken so as to ensure that the entire research team had a proper understanding of the JBI. We refer to this activity as an informal domain analysis.

The second activity that was conducted outside the scope of the basic evaluation of the Willow architecture was a security analysis of a typical JBI architecture using probabilistic risk analysis. Clearly security is important to the JBI concept but the security requirements are somewhat unique. The purpose of using probabilistic risk analysis was to try to determine in as much detail as possible what the threats and vulnerabilities will be for a JBI.

The majority of the technical results of this project are documented in papers and reports. These papers and reports are listed in section 5, and their detailed content is not duplicated here. In this report we include only a summary of the major results. We do, however, include three appendices to this report that are technical reports which have not been published elsewhere:

- Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems
- Domain Analysis of the Joint Battlespace Infosphere
- The Application of Probabilistic Risk Analysis to Security

This report is organized as follows. Section 2 provides a summary of the Willow project together with a discussion of the activities undertaken to apply it to the JBI in order to provide the necessary background. A summary of the research results is presented in section 3. Section 4 is a list of personnel associated with the project. Publications to date resulting from this grant are listed in section 5. Presentations given are listed in section 6, and discoveries, patents and inventions in section 7. Copies of the technical reports are included as appendices A, B, and C.

2. PROJECT DESCRIPTION

2.1. Introduction

As a society, we are becoming increasingly dependent on the continuous, proper functioning of large-scale, heterogeneous, distributed information systems. These systems are formed from large numbers of components, both hardware and software, originating from multiple sources assembled into complex and evolving structures spread across wide geographic areas. Our goal is to develop techniques that help to either prevent disruptions or ensure that these systems can continue to provide acceptable though not necessarily complete levels of service, that is to *survive*, in the face of serious disruptions to their normal operation. In this section we present the main features of the *Willow architecture*, an architecture that provides a comprehensive architectural approach to survivability [14] in critical distributed applications. The important contribution of the architecture is the merging of three major approaches to dealing with faults into a single system. Although this section only summarizes the architecture, other details are available in other cited papers.

The Willow architecture is based on the notion that survivability of a distributed application requires a broad approach to dealing with faults in the application, an approach that includes fault avoidance, fault elimination, and fault tolerance. Thus it includes mechanisms: (a) to *avoid* the introduction of faults into the systems at the time of initial deployment or subsequent enhancement; (b) to *eliminate* (i.e., remove) faults from a deployed application once they are either identified or merely suspected but before they can cause failure; and (c) to *tolerate* the effects of faults during operation. These various mechanisms are all based on a general notion of *reconfiguration* at both the system and the application levels together with a framework that implements a monitor/analyze/respond approach to the identification and treatment of faults.

Experience has shown that serious faults are often introduced during system deployment and enhancement. For example, software components are distributed with default passwords set, the wrong software version is deployed, corrective patches are not applied, and so on. Prevention and repair of such faults are the reason that fault avoidance and fault elimination are needed. A novel special case of fault elimination occurs in circumstances where a fault is suspected but not diagnosed. In such cases, it is desirable for the component(s) with the fault to be removed from the system if circumstances indicate that the fault might be manifested. As an example, consider worm attacks against servers that have been vulnerable to worms in the past. If there is a possibility that some servers might still be vulnerable and a new worm is detected, rapidly disconnecting all the servers in the suspect class from the network until the worm is eliminated might be a prudent precaution. In this case, a fault is suspected and eliminated temporarily, although during the period that the fault is eliminated, functionality probably had to be changed. This particular form of fault elimination is sometimes referred to as *posturing*.

Although fault avoidance, fault elimination, and fault tolerance are related but different concepts, each is an approach to dealing with faults, and all three are provided by Willow. Their implementations share a number of architectural elements and a common architectural theme of reconfiguration. Despite all three being designed to deal with faults, they are not entirely compatible and actions taken by one can conflict with actions taken by another. For example, if a fault such as a

widespread power failure occurs while an activity is underway to eliminate a fault by replacing software components, it might be necessary to reconfigure the system to deal with the power loss as a higher priority than the ongoing software replacement. The Willow architecture contains mechanisms to deal with such conflicts.

Security of the Willow architecture and its data sources is of paramount importance because, if it were compromised, an intruder could do immense damage. Various techniques have been developed to support the Willow architecture and are described.

2.2. Critical distributed applications and survivability

The distributed information systems for which the Willow system is being developed arise in the context of the nation's critical infrastructures. Transportation, telecommunications, power distribution and financial services are examples, and such services have become essential to the normal activities of society. Similarly, systems such as the Defense Department's Global Command and Control System (GCCS) and Joint Battlespace Infosphere (JBI) are essential to the nation's defense operations.

In such a system, all or most of the service it provides can be lost quickly if certain faults arise in its information system. Since societal and military dependence on critical infrastructures is considerable, substantial concern about the dependability of the underlying information systems has been expressed, particularly their security [17, 19].

Critical information systems, whether civilian or military, are typically networks with very large numbers of heterogeneous nodes that are distributed over wide geographic areas [13]. It is usually the case that these systems employ commodity hardware, and that they are based on COTS and legacy software. The networks typically implement either: (a) point-to-point connectivity; (b) full packet-switched connectivity similar to the Internet; or (c) specialized connectivity such as publish/subscribe. Irrespective of the connectivity, there are different numbers of the different types of node, and the different types of node provide different functionality. Some nodes provide functionality that is far more critical than others leading to a situation where the vulnerabilities of the information system tend to be associated with the more critical nodes.

An important architectural characteristic of many critical information systems is that provision of service to the end user frequently involves several nodes operating in sequence with each supplying only part of the overall functionality. This *composition of function* can be seen easily in any command-and-control system where command hierarchies and associated information databases are used.

The Willow architecture is designed to enhance the survivability of critical information systems. Information-system survivability has been defined in detail elsewhere [14], and so we merely review the meaning of the term here. Informally, the concept is: (1) an information system should provide its complete functionality for some, possibly pre-specified, fraction of the time; (2) the system should meet pre-defined reduced or *different* requirements if it cannot provide full functionality because of failures (including security attacks); and (3) several sets of different requirements might be stated to permit useful if limited functionality to be specified to accommodate different degrees of damage. Given that failures are inevitable, it is essential that attention be paid

to how failures are handled and, in particular, how the systems' users will be served during these times. Note that survivability is not a different name for the existing term "graceful degradation". The essential distinctions are: (1) the possibility of different rather than merely reduced functionality; and (2) the precise statement of these different functionalities in user-defined requirements.

2.3. Types of fault

An important aspect of the Willow architecture is its goal of dealing with certain types of very elaborate faults. This is in contrast to typical systems that implement fault tolerance where the assumption is usually made that only one component at a time will be subject to a fault. In this section we review the types of fault that the Willow architecture is being designed to tolerate. All of these types of fault are important for the JBI.

Non-local faults

Traditional fault avoidance and fault elimination techniques such as programming languages with strong type checking and systematic inspection can be used in the development of individual components for critical information systems. Similarly, to effect component- or node-level fault tolerance designers can employ traditional techniques such as N-modular redundancy in processing, communications, data storage, power supplies, and so on. These techniques cope well with a wide variety of faults but in this research we are not concerned with faults that affect a single hardware or software component or even a complete application node. We refer to such faults as *local*, and, although they are important, we assume that local faults are dealt with to the extent possible by existing techniques.

We are concerned with the need to deal with faults that affect significant fractions of a network, faults that we refer to as *non-local*. Some examples of non-local faults are: (1) several nodes having software components lacking essential security patches, defective virus databases, or improperly configured operating systems; (2) extensive loss of hardware or a widespread power failure; (3) failure of an application program; (4) coordinated security attacks, and so forth. Non-local faults are much more difficult to deal with than local faults for obvious reasons.

Complex faults

In critical information systems, non-local faults are only a small part of the problem. The scale of modern critical applications raises the following three additional issues:

- *Fault Sequences*

It is necessary to deal with fault *sequences*, i.e., new faults arising while earlier faults are being dealt with. The reason that this is a serious complication is that responses to faults in some cases will have to be determined by the overall application state at the time that the fault is manifested.

- *Fault Hierarchies*

Any non-local fault must be dealt with appropriately. However, once the effects of a fault are detected, the situation might deteriorate or more information might be obtained leading to the subsequent diagnosis of a more serious fault requiring more extensive action. This suggests

the notion of a fault hierarchy and any approach to fault elimination or fault tolerance must take this into account.

- *Interdependent Application Faults*

Many critical infrastructure applications depend upon one another. For example, the nation's financial system depends upon commercial electric sources and upon commercial telecommunications facilities. Faults and the resulting losses of service in the electrical network could cause losses of financial services if electric supply were interrupted for protracted periods even though nothing was wrong with the financial system itself. In the case of the JBI, the interdependence would be between multiple JBIs or between a JBI and some separate information supply network that was acting as a publisher.

2.4. Willow concepts

Our approach to dealing with complex, non-local faults is to use an extended form of a particular system architecture that is referred to as a *survivability architecture* or an *information survivability control system* [20]. A survivability architecture is characterized by having as its basic structure a control loop that operates during system execution to monitor the system, analyze it, and effect some form of response if analysis reveals a problem. This monitor/analyze/respond structure is the basis of many network intrusion detection systems, for example [18].

The Willow architecture generalizes the control concept and includes multiple inter-operating but separate loops. In addition, during the execution of the critical application, the state of its *operating environment* is monitored along with the state of the application itself, and the resulting integrated set of state information is analyzed. Necessary changes to the critical application system are effected as required. In the case of the Willow architecture, all of these notions are applied in a very general sense. Application system state, for example, includes the detailed software configurations in use on the various nodes. The state of the operating environment includes intelligence information about security threats together with details about new releases of the software components used within the system. Finally, necessary changes include correcting defective operating system or application configurations.

Changes to the critical application system include both software updates that will be transparent to the system's user and modifications to the system's functionality that will not be transparent. Functionality changes include reducing some services, ceasing others, and perhaps initiating application services that are not normally available (such as basic emergency services). Monitoring and change are carried out by sensing and actuating software that resides on network elements of interest. Analysis is performed by servers that are not part of the application network, and communication between the monitored nodes and the servers is by independent communications channels. The Willow architecture concept including multiple instances of the basic control loop is illustrated in Figure 1.

The necessary changes in a critical application are effected by *reconfiguration*. The Willow architecture supports reconfiguration in a very broad sense, and reconfiguration in this context refers to any scenario that is outside of ongoing, "steady-state" operation. Thus, for example, initial system deployment is included intentionally in this definition as are system enhancements and upgrades, posturing, recovery from hardware failure, and so on. All are viewed merely as special cases of

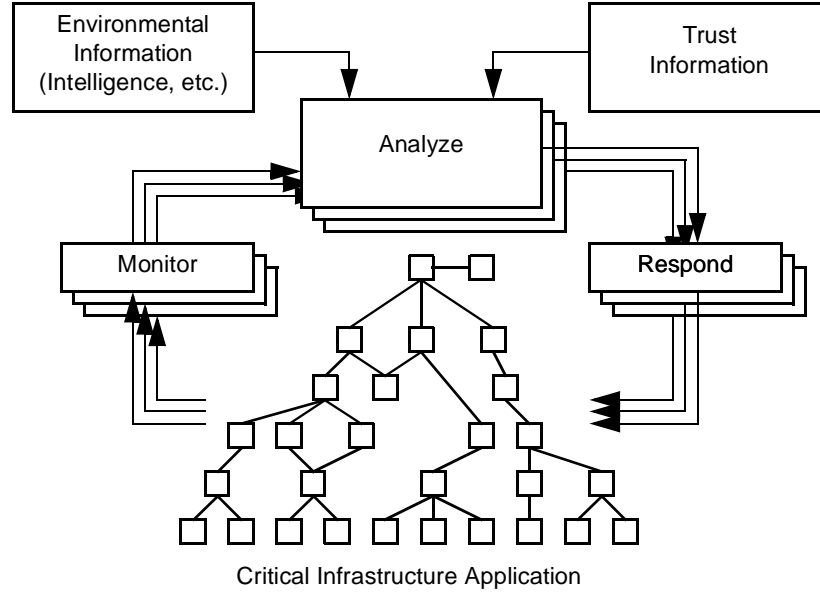


Figure 1. Willow survivability architecture concept

the general notion of reconfiguration. We refer to reconfiguration that is effected before the system has sustained any damage as *proactive* and after the system has sustained damage as *reactive*. Proactive reconfiguration adds, removes, and replaces components and interconnections, or changes their mode of operation. In a complementary fashion, reactive reconfiguration adds, removes, and replaces components and interconnections to restore the integrity of a system in bounded time once damage or intrusions have taken place. Examples of specific system reconfigurations that can be supported by Willow are:

- Application and operating system updates including component replacement and re-parameterization. Initial application system deployment is treated as a special case of this.
- Planned posture changes in response to anticipated threats.
- Planned fault tolerance in response to anticipated component failures.
- Systematic best efforts to deal with unanticipated failures.

Since reconfiguration could be used as a means of security attack, the input that is used in the Willow decision-making process is managed by a comprehensive trust mechanism [5]. In addition, extensive protection is used for the elements of the Willow architecture itself.

The Willow concept derives from a realization that system and application configuration control and application fault tolerance are two different aspects of the general problem of overall control of distributed systems. Both utilize specialized knowledge about the applications, the resources available and the application state to prepare and react to changing conditions for distributed applications. The difference lies in the time frames at which the two aspects operate, and in the

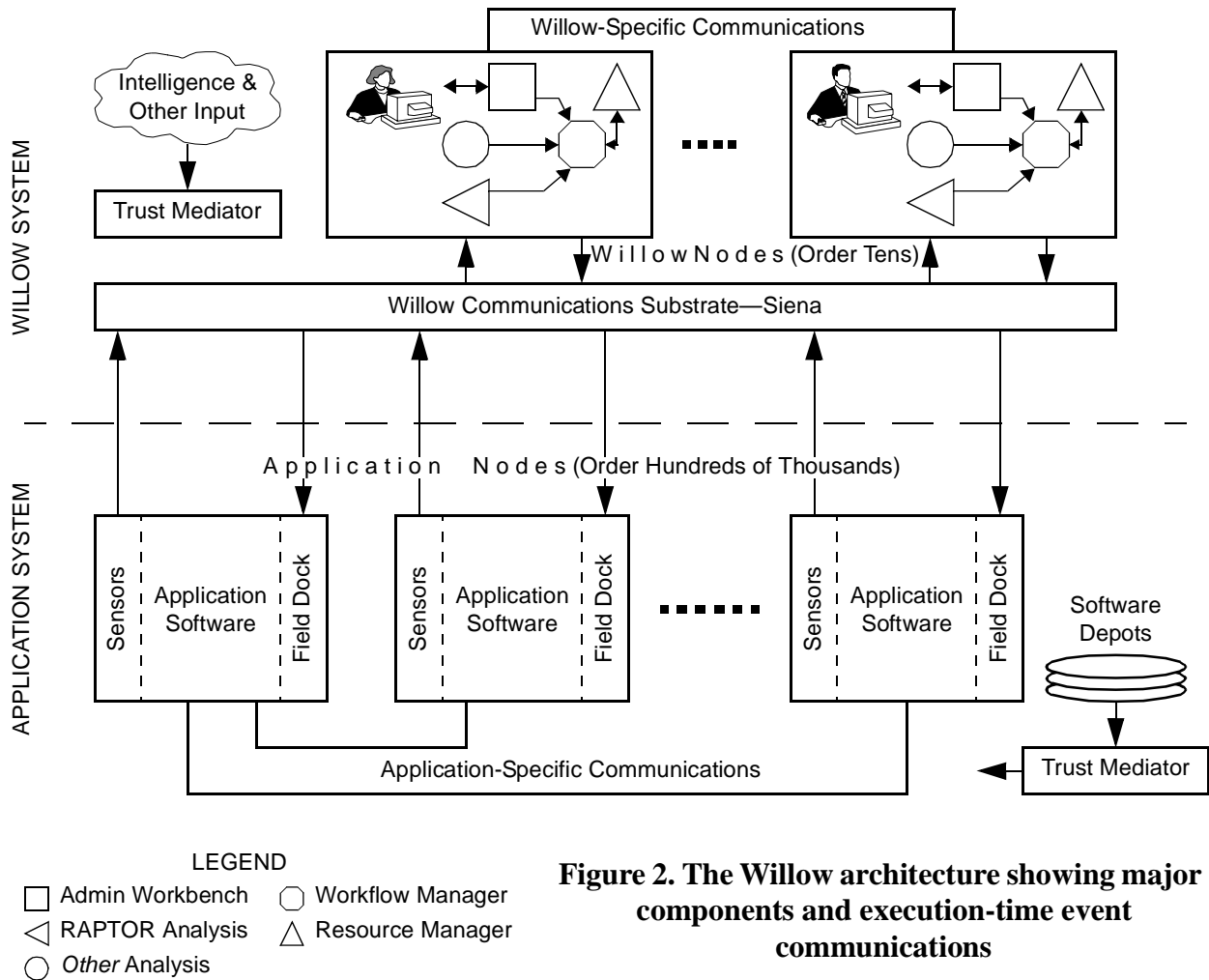


Figure 2. The Willow architecture showing major components and execution-time event communications

mechanisms used to detect and respond to circumstances needing action. Application fault tolerance is mostly time-bounded, needing prescribed responses to anticipated faults. Software configuration management involves run-time analysis of application state to determine necessary basic actions from a series of prescribed facts and newly available application elements (new software versions, operating system conditions, etc.)

2.5. The Willow architecture

Control loops

The major components and the major forms of communication in the Willow architecture are illustrated in Figure 2. The top part of the figure shows the majority of the Willow system and the lower part shows the distributed application that Willow is supporting. In the case of this project, the application would be a JBI. In general, Willow assumes the distributed application operates on a traditional network and has computation and communication requirements that are met by an unspecified (but assumed large) set of nodes together with appropriate communications links. The Willow system operates (at least in this example) on a separate network, the details of this network will depend upon the computational and communication needs of the spe-

cific Willow system for the application.

The fundamental structure of the Willow architecture, the set of *control loops*, has sensing, diagnosis, synthesis, coordination, and actuation components and these are depicted in Figure 2. The control loops begin with a shared *sensing* capability shown within the application nodes. Sensors can include reports from applications, application heartbeat monitors, intrusion detection alarms, or any other means of measuring actual application properties.

From sensing events, independent *diagnosis and synthesis* components build models of application state and determine required application state changes. In the current Willow architecture, there are two of these components—the Administrator’s Workbench for proactive reconfiguration and RAPTOR for reactive reconfiguration. Additional diagnosis and synthesis components can be added easily and this is illustrated in the figure by the “*Other*” analysis component.

Synthesis components issue their intended application changes as *workflow* requests. These are coordinated by the workflow and the resource managers to ensure that changes occur correctly and smoothly within the application.

When workflows are allowed to activate, workflow events are received by the Field Docks located within the application nodes and result in local system state changes. The Field Dock infrastructure provides a single standard interface for *universal actuation* at application nodes [8, 9, 10]. Actuation completes the control loop cycle.

Proactive control

The proactive controller, the Administrative Workbench, is an interactive application allowing system administrators to monitor system conditions remotely, adjust system properties, and most importantly, cause the propagation and implementation of proactive reconfigurations.

The Software Depot in Figure 2 represents an external source of information that may be required by Willow in order to complete its reconfigurations. In general, there will likely be many such depots. This information may include models of new applications, components needed in a new configuration (but which are not already available locally on the affected application node), and components that provide additional actuators to, for example, access built-in reconfiguration capabilities for specific applications.

Reactive control

The reactive controller, known as RAPTOR, is a fully automatic structure that is organized as a set of *finite state machines*. The detection of the erroneous state associated with a fault (i.e., error detection) is carried out by a state machine because an erroneous state is just an application system state of interest. As the effects of a fault manifest themselves, the state changes. The changes become input to the state machine in the form of events, and the state machine signals an error if it enters a state designated as erroneous. The various states of interest are described using predicates on sets that define part of the overall state. The general form for the specification of an erroneous state, therefore, is a collection of set definitions that identify the application objects of concern and predicates using those sets to describe the various states for which either action

needs to be taken or which could lead to states for which action needs to be taken.

In an operating application such as a JBI, events occurring at the level of individual application nodes are recognized by a finite-state machine at what amounts to the lowest level of the system. This is adequate, for example, for a fault like a wide-area power failure that was caused by some form of battle damage. Dealing with such a fault might require no action if the number of affected nodes is below some threshold. Above that threshold might require certain critical application nodes to respond by limiting their activities. As node power failures are reported so a predefined set, say *nodes_without_power*, is modified, and once its cardinality passes the threshold, the recognizer moves to an error state. A JBI might respond to a wide-area loss of power by limiting the amount of material that can be published, limiting the allowable set of subscriptions, and so on.

The notion of a fault hierarchy requires that more complex fault circumstances be recognized. A set of nodes losing power in one part of a JBI is one fault, a set losing power in another part is a second, but both occurring in close temporal proximity might have to be defined as a separate, third fault of much more significance because it might indicate a coordinated enemy or terrorist attack. This idea is dealt with by a *hierarchy* of finite-state machines. Compound events can be passed up (and down) the hierarchy, so that a collection of local events can be recognized at the regional level as a regional event, regional events could be passed up further to recognize national events, and so on. As an example, a widespread coordinated security attack might be defined to be an attack within some short period of time on several collections of nodes, each within a separate administrative domain. Detection of such a situation requires that individual nodes recognize the circumstances of an attack, groups of nodes collect events from multiple low-level nodes to recognize a wide-area problem, and the variety of wide-area problems along with their simultaneity recognized as a coordinated attack.

Dealing with conflicting goals

During operation of a distributed application in a Willow system, reconfiguration could be initiated to avoid a fault (posturing), to eliminate a fault (classical reconfiguration in which some set of software component are updated in some way to correct a problem), or to tolerate a fault (physical damage, security attack, etc.). Thus there are multiple sources of initiation for reconfiguration and they are asynchronous. Clearly, this means that more than one might be initiated at the same time in which case either one has to be suspended or there has to be a determination that they do not interfere. Worse, however, is the prospect that one or more new initiations might occur while already initiated reconfigurations are underway. Some reconfigurations are more important than others—tolerating a security attack once detected, for example, is more important than eliminating a minor fault in some piece of application software—and so reconfigurations underway might have to be suspended or even reversed. Unless some sort of comprehensive control is exercised, the system would quickly degenerate into an inconsistent state.

One approach would be to have each source make its own determination of what it should do. The complexity of this approach makes it infeasible. An implementation would have to cope with on-the-fly determination of state and, since initiation is asynchronous, that determination would require resource locking and so on across the network. The approach we have taken is to route all requests for reconfiguration through a resource manager/priority enforcer called ANDREA.

The prototype ANDREA implementation uses predefined prioritization of reconfiguration requests and dynamic resource management to determine an appropriate execution order for reconfiguration requests using a distributed workflow model to represent reconfiguration requests. The workflow model formally represents the intentions of a reconfiguration request, the temporal ordering required in its operation, and its resource usage. Combined with a specified resource model, this information is the input to a distributed scheduling algorithm that produces and then executes a partial order for all reconfiguration tasks in the network.

Scheduling within ANDREA is preemptive allowing incoming tasks to usurp resources from others if necessary so that more important activities can override less important ones. Transactional semantics allow preempted or failed activities to support rollback or failure, depending on the capabilities of the actuators enacting the reconfiguration. ANDREA supports an event-driven interface so that adaptive systems can be easily interface and observe Willow's reconfiguration protocol.

The current ANDREA system demonstrates formal specification of complex reconfiguration tasks across multiple nodes of a distributed application system. Redundant application of complex, yet menial management tasks is completely automated. Multiple administrators and automated controllers can engage in proactive and reactive management of the system without resource conflicts and without the potential of interrupting more important tasks initiated by another controller. A controller that initiates a high priority task is guaranteed to receive resources in bounded time given that actuators for current tasks are compliant with interruptible transaction semantics. In general, the current ANDREA system demonstrates that asynchronous parallel control-loops can interact to enhance, rather than degrade, the survivability of an application system.

Communication

The communication challenges presented by the Willow architecture are considerable—sensor information has to be transmitted from nodes to the analysis components of the control loops, reconfiguration commands have to be transmitted from the analysis components to the nodes, replacement software components have to be transmitted from Software Depots, and so on. This would not be a significant challenge but for the fact that one of the targets of the Willow architecture is very large networks, i.e., networks with many nodes.

The approach to communication that Willow takes is to reduce as much of the communication as possible to event notification and then to use a highly efficient wide-area event notification service as the communications substrate. The specific event notification service that Willow uses is Siena [4]. Siena uses a broad range of techniques to maximize the efficiency of its communication service including: (1) a high-level router structure to link its various servers; and (2) routing only a single copy of a notification as far as possible, only replicating it when necessary. Siena avoids explicitly a solution based on a central database since such an approach would not scale to the degree that Willow requires and would severely limit applicability.

The major communication service that cannot be reduced to event notification is the transmission of files associated with software or bulk data distribution. This is handled by using events to define and initiated file transfers and then allowing the actual transfers from Software Depots to

take place via other network communications services.

Security

Given the pervasive notion of control in the architecture, there is the risk that an adversary might exploit the Willow infrastructure itself to cripple the distributed application. The defensive measures in the Willow architecture deal with two major concerns. The first is securing the mechanisms of the architecture itself, and the second is ensuring that the information used by the system is current, accurate, and intact. Thus we defend against two types of adversaries—those that seek to directly subvert the control mechanisms, and those who try to do that indirectly by tampering with the data used by the control mechanisms.

Securing the mechanisms of the architecture breaks down into the following three technical problems:

- Protecting the Willow servers that conduct the analysis.
- Protecting the sensors and actuators that reside on application nodes.
- Protecting the communication between the various Willow components.

The first of these three problems can be solved using conventional techniques such as physical security, personnel authentication (via passwords, biometrics, etc.), and cryptography.

The second problem is much harder to solve. Sensors and actuators operate on a large, heterogeneous collection of nodes located in a variety of administrative domains. We must, therefore, require that trusted components (the sensors and the actuators) operate correctly and continuously on untrustworthy hosts (the application nodes). If the sensor code could be either tampered with or replaced, for example, an adversary could arrange for the transmission of erroneous data for analysis and thereby force malicious control actions.

Protecting the trusted components is achieved using a combination of several approaches. The possibility of an adversary understanding the software by analyzing the binary statically is reduced by code obfuscation. Tamper resistance is further improved by using randomization in the observable behavior of software, and by employing temporal and spatial diversity of the trusted code. By temporal diversity we mean the intentional replacement of the software at periodic interval, with a new version that has different observable behavior. A randomization technique [23] is used to produce the various versions of the software such that only trusted components know the randomization that was used. By spatial diversity we mean the use of intentionally diverse versions of the software whose functionality is nominally identical (the sensors, for example) throughout the system. Production of the diverse versions can be achieved using specially modified compilers that generate the necessary diversity.

The third problem in the list above (securing the communication between the various Willow components) is solved partially by encryption but a new challenge is raised by the Willow architecture—securing the publish/subscribe communication service that underlies the infrastructure [4]. For Willow, the wide-area publish/subscribe service must handle information dissemination

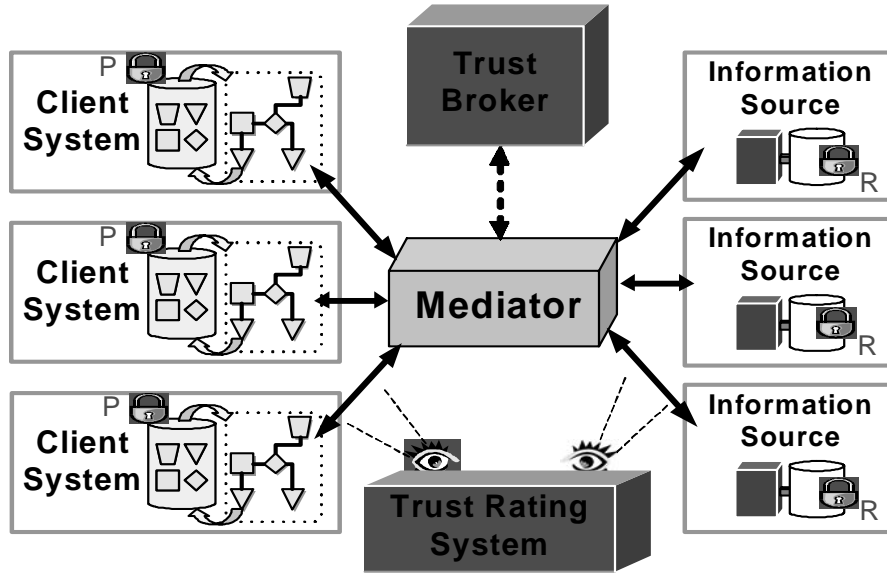


Figure 3. Organization of the Trust Broker mechanism

across distinct authoritative domains, heterogeneous platforms, and a large, dynamic population of publishers and subscribers. Such an environment raises serious security concerns. The general security needs of the client of a publish/subscribe service include confidentiality, integrity, and availability, while the security concerns of the publish/subscribe infrastructure itself focus primarily on system integrity and availability. A detailed discussion of how some aspects of the problem can be solved is available in a separate publication [23].

The second broad category of security issues (security services to allow the latest, most accurate and most trusted information to be used) is effected by a secure mediator infrastructure (SMI). There are two design goals for the SMI. First, administrators can precisely control and coordinate where and how data are obtained, via a centralized *trust broker* that gathers and disperses meta-data on the trustworthiness of data sources. Second, the SMI allows authentic data distribution [5] without the use of on-line signing keys. Keyless hashing primitives are used to validate the distributed data, with the data signed infrequently using off-line keys. By avoiding on-line keys, which might be stolen by an attacker, we enhance security and reduce the administrative burden. Finally, data mediation allows translation of data from different sources, and selection of data sources based on different criteria over the trust meta-data. This SMI design provides scalability, delegation and separation of concerns, so that the Willow infrastructure can obtain the best information from the widest possible set of sources with good security and low administrative overhead. The high-level organization of the Trust Broker is shown in Figure 3.

2.6. Network Simulation System

Experimentation is a crucial element of research with projects like Willow. Unfortunately, several factors present serious impediments to experimentation. In general, large networked information systems such as a JBI are enormous in physical scale, cost and complexity, and this makes it infeasible to replicate them in the laboratory.

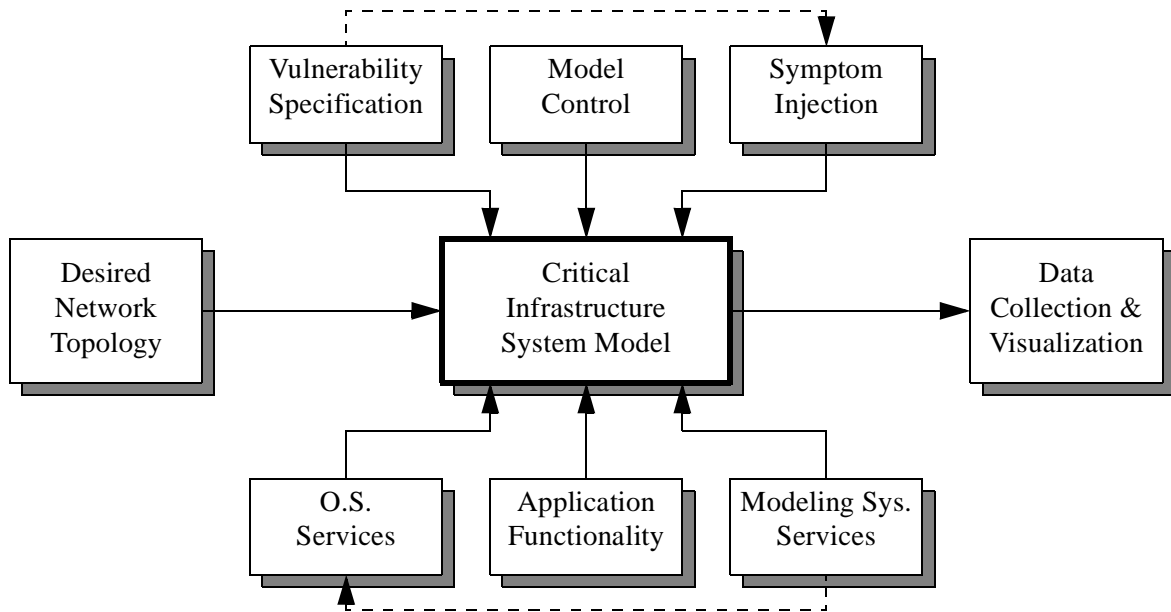


Figure 4. The RAPTOR modeling system architecture

Given the impediments to direct experimentation with real infrastructures, we have adopted an experimental approach based on operational models. Building operational models of critical infrastructure information systems presents two significant challenges: (1) modeling the critical but no other aspects of infrastructure systems with sufficient accuracy and completeness; and (2) facilitating inclusion in the model of relevant architectural mechanisms to be developed or evaluated. For purposes of experimentation, an operational model has to represent relevant functional and architectural features of a given system, as well as its operational environment, including a dynamic model of internal failures and external threats. Once such a model is built, mechanisms must be present to allow prototypes of architectural survivability mechanisms to be introduced. Both models and architectural supplements must be instrumented for collection of data needed to analyze and evaluate survivability mechanisms.

The system we have developed to meet the various requirements outlined above is called RAPTOR. For purposes of experimentation, the RAPTOR system provides the user with an efficient, easily manipulated operational model of a distributed application with extensive control, monitoring, and display facilities. Figure 4 provides an overview of the system.

A RAPTOR model is specified by defining the desired *topology* and the desired *application functionality*. From the topology, the model is created using services from the modeling system's support libraries and using application software provided by the model builder. *Vulnerabilities* to which the model should be subject are defined and controlled by a user-defined vulnerability specification. During the execution of a model, *symptoms* can be injected into the model to indicate any event of interest to the user. Events might include security penetrations, hardware failures, etc. Any *data* of interest to the user can be collected and made available to a separate process

(possibly on a remote computer) for *display* and analysis. Finally, since multiple independent models can be defined from separate topology specifications, complex systems of *interdependent* critical networks, such as a set of interoperating JBIs, can be modeled (see Figure 5).

The basic semantics of a model is a set of concurrent message-passing entities that we refer to as *virtual message processors*. Figure 6 depicts the general structure of a virtual message processor. A virtual message processor is provided with a queue of incoming messages that it can read and process as it chooses. Usually, these messages are routed from the input queue to programmable message interpreters. Any new messages generated as a result of interpreting received messages are sent immediately although their arrival times at their destinations can be controlled. Messages can be generated asynchronously also if needed based on, for example, a timer event.

Within the modeling system, a network node is modeled as a set of one or more virtual message processors each of which is executed by a separate OS-level thread. A typical simple node can be modeled with a single virtual message processor and hence a single thread. More complex nodes can be modeled as a collection of threads thereby allowing such nodes to exhibit concurrent internal behavior. The use of threads for modeling nodes allows the model of a single node to be itself concurrent thereby permitting issues such as synchronization errors and race conditions to be modeled realistically.

Node-to-node communication is modeled by message passing between threads. Messages are passed through memory and so message passing is very efficient. Any thread can send a message to any other thread subject to restrictions imposed by a model's topology (see below). In order to permit models in which resource contention might occur, the input message queue for a virtual message processor can be given a maximum size. If the input message queue is full when a message arrives, the attempt to send the message to that virtual message processor fails. Message transmission times can also be specified to allow transmission delays to be modeled.

Since a JBI is a critical networked information system, we determined the utility of the RAPTOR modeling system to the JBI technology by developing a simulation model of a large JBI. The model implemented the key elements of the Siena publish/subscribe routing algorithms. A JBI model with several hundred routers was created and exercised with large numbers of random publications and subscriptions. Based on this simple feasibility study, we concluded that the capabili-

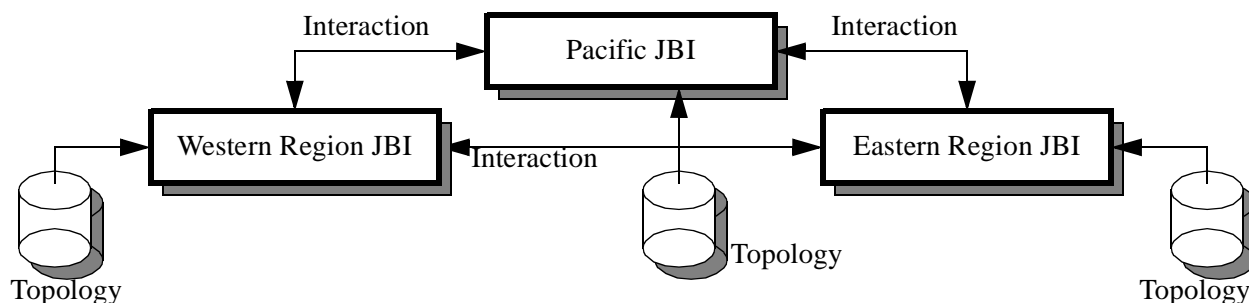


Figure 5. A Model of Multiple Interacting Infrastructure Systems

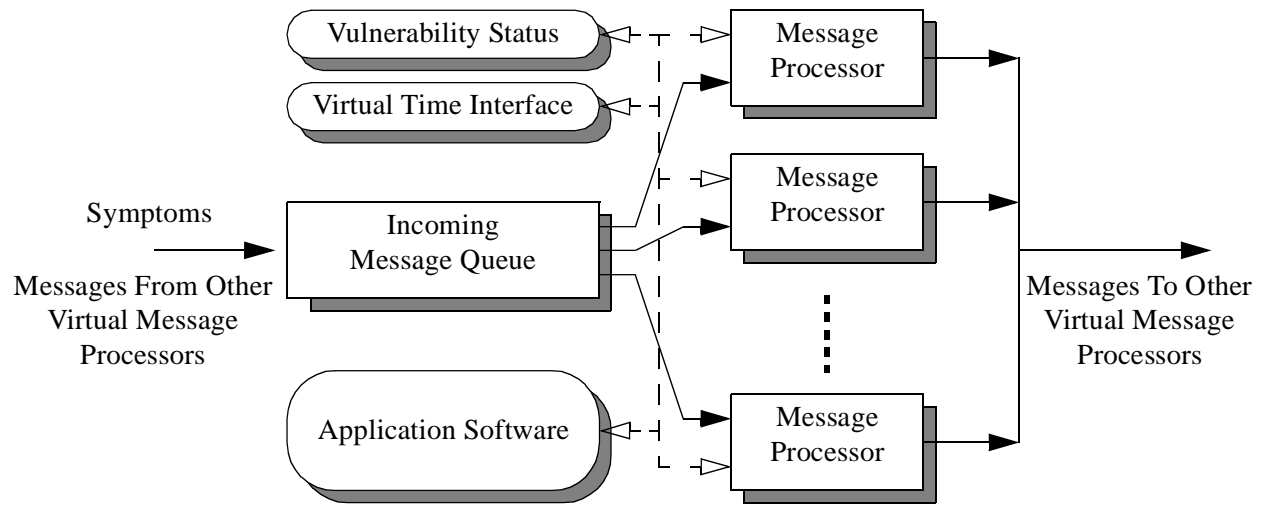


Figure 6. RAPTOR Virtual Message Processor

ties that the RAPTOR simulation system provides are quite adequate to model performance and survivability of large JBI systems.

2.7. Overall Evaluation

The Willow architecture is designed to provide a sophisticated service to civilian and military critical applications operating on large and complex networks. Evaluating the architecture is difficult because of the scales of the parameters involved. Typically, the numbers of nodes in a network will be large, there might be several node types, the functions provided by the application will be sophisticated and there will be many of them, the number of users of the system will be large, and data maintained in databases will generally be extensive. For the JBI, evaluation faces all these challenges.

Ideally, experimental evaluation of the Willow architecture would be conducted on a full-scale implementation that was supporting representative applications with the system characteristics just noted. Clearly this is infeasible in general, but for the JBI concept it is especially difficult because there is no operational, full-scale JBI. Given this, another approach has to be taken. Our approach to evaluation is in two parts. In the first part, the major elements of the architecture have been studied individually using a combination of analysis, small-scale implementation, and simulation. In the second part, a laboratory-scale Willow system that supports a prototype JBI implementation has been developed as a case study and to evaluate feasibility. In this section, we summarize briefly the various evaluation efforts of the major elements, and review the laboratory-scale JBI case study.

Evaluation and status of components

The reconfiguration component of the Willow architecture is derived from the Software Dock system [10]. A prototype Software Dock was implemented as part of a previous research project and its performance evaluated.

The proactive control component—the Administrator’s Workbench—has been developed as a layer on top of the existing Software Dock. It connects to the internal event communication mechanism used by the Software Dock to provide network level control of one or more Field Docks. Since its operation is directly tied to that of the Software Dock, we expect that the Software Dock performance carries over to the Workbench.

The RAPTOR reactive reconfiguration mechanism has been evaluated previously using two models of critical infrastructure applications. The first model was of the nation’s financial system and the second was of the nation’s electric power grid. In both cases, models were built that implemented abstract versions of the application, including only essential functionality, along with a complete implementation of the error detection and error recovery mechanisms. Both models were of realistic size, each involving thousands of application nodes. These two models were executed using a network simulation system and faults injected. Details of the results can be found elsewhere [6] but the overall conclusion was that the RAPTOR mechanism as modified to operate in the Willow system will provide the desired fault-tolerance capability.

The ANDREA workflow system is the only major component of the Willow architecture that cannot be evaluated easily outside the context of a full-scale implementation. Its performance in real time is the most important aspect of evaluation but this is difficult to measure because it depends on so many elements of a complete implementation. Informal assessment of the performance of the preliminary implementation has been completed and demonstrated feasibility. The current ANDREA system allows scaling of compact workflow specifications to large-scale networks because it implements run-time, multicast binding of workflow commands to actuation sites rather than requiring explicit location specification.

Many aspects of Siena, our wide-area event notification system, have been studied. A prototype implementation has been built and used in a number of trials by ourselves and by others who have downloaded the implementation. In addition, we have carried out extensive simulation studies to assess how the system performs in terms of scale, throughout, etc. Detailed reports of these assessments have been reported by Carzaniga et al [4]. We conclude from these studies that the anticipated workload imposed upon Siena by the expected operating circumstances of a production Willow implementation supporting a JBI implementation are well within its capacity provided the required time bounds on response dictated by the JBI application are within the scope of the system.

JBI Case studies

Two prototype Willow systems, each with a JBI implementation, have been developed to determine the applicability of various aspects of the Willow architecture to the JBI. We refer to them here as System 1 and System 2. System 1 was an initial feasibility case study. System 2 was developed after System 1 so as to benefit from the initial experiments and to explore issues closer to realistic sized JBIs.

System 1 implemented all the different aspects of the Willow architecture except certain aspects of the security mechanism. The system included an administrator’s workbench, a RAPTOR error detection mechanism, and a communications infrastructure based on the Siena publish/subscribe

system. The JBI implementation in System 1 was based on Siena and so this feasibility study implementation used two entirely separate publish/subscribe systems—one provided the Willow architecture’s own essential communication and the second implemented the core functionality of our JBI.

A production JBI system will require proper initial deployment, configuration and maintenance of all of its software elements, and will have to be updated quickly and efficiently if defects in the system are observed. In addition, a JBI will be an attractive target to an adversary for many reasons. Such a system might be attacked in various ways by hackers or disabled by battle damage, physical terrorism, software faults, and so on. It is essential, therefore, that a JBI be survivable, and the Willow architecture is a candidate implementation platform.

The Siena-based JBI implementation in System 1 included several information-processing modules (known as fuselets) and synthetic publishers and subscribers. All of the components of the system were enhanced to allow them to respond appropriately to reconfiguration actions. In addition, the different elements of the implementation were extended deliberately with vulnerabilities so as to permit demonstration and evaluation of the reconfiguration capabilities of Willow.

Operating on the Willow architecture, the System 1 JBI implementation was subjected to a preliminary evaluation by fault injection. The initial deployment of the system to a test network was entirely automatic, and the system was shown to adopt new postures under operator control as desired. System 1 was also shown to reconfigure automatically when network faults were injected.

System 2 was a major refinement of System 1 designed to implement all the necessary technology required for a fully scalable system. To provide a trustworthy target, the computers used as the platform were configured to use the Immunix operating system. This is a specially-modified commercial version of Linux that includes a number of mechanisms which ensure the absence of common vulnerabilities. In addition, the Immunix systems were deliberately configured to provide the bare minimum service so as to further limit their potential vulnerabilities. These machines were used to execute the publish/subscribe routers needed to support the JBI implementation. The JBI clients were executed on standard MS Windows machines.

The JBI implementation in System 2, shown in Figure 7, was much more elaborate than the one built for System 1. The major differences are: (1) the inclusion of databases into the system to hold data that could not be communicated efficiently using the Siena event-notification mechanism; (2) a highly functional observation and command interface client; and (3) support for arbitrary numbers of publishers and subscribers. This JBI was designed to be a preliminary implementation of the JBI specified for the DARPA OASIS II project. It supports publication of sensing information about aircraft locations, publication of weather information, publication of air tasking orders, and subscription to all of these published information streams.

The Willow implementation in System 2 implemented many new mechanisms including:

- A new version of the survivability specification language and a new translator of survivability specifications. The new language provides more general support for survivability and support

for temporal specifications.

- A new communications mechanism known as *site-select addressing* that permits the major elements of the reactive control mechanism to be implemented without requiring that extensive amounts of network state be maintained by the Willow system.
- A new actuation mechanism based on an agent structure in which a hierarchy of agents interacts with the networked application in a way that associates actuation semantics with appropriate levels in the application.
- A new mechanism termed *result harvest* that permits publishers in publish/subscribe systems to receive replies from subscribers in efficient ways.

2.8. Related Work

All of the component technologies used in the Willow architecture have been the subject of research although no research has been conducted with the our goal of integrating the components to provide a comprehensive approach to dealing with faults.

Software deployment and configuration management have been studied by numerous researchers. Detailed information on related technologies has been presented by Hall [7]. System-level approaches to fault tolerance are described by several authors [1, 3, 2]. While these efforts are significant in their own right, they do not address non-local faults, and they tend not to address systems as large and complex as critical informations systems.

Reconfiguration has also been studied by others in systems such as CONIC and Darwin [15, 16]. Purtilo and Hofmeister studied the types of reconfiguration possible within applications and the requirements for supporting reconfiguration [11]. Details of related work on event notification services can be found in the paper by Carzaniga et al [4].

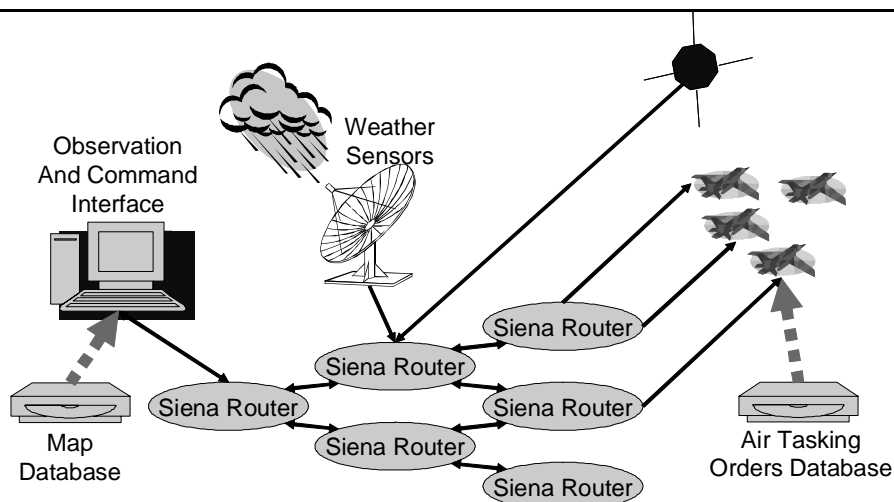


Figure 7. System 2 JBI Implementation

2.9. References for This Section

1. Alvisi, L. and K. Marzullo. "WAFT: Support for Fault-Tolerance in Wide-Area Object Oriented Systems," Proceedings of the 2nd Information Survivability Workshop, IEEE Computer Society Press, Los Alamitos, CA, October 1998, pp. 5-10.
2. Bhatti, N., M. Hiltunen, R. Schlichting, and W. Chiu. "Coyote: A System for Constructing Fine-Grain Configurable Communication Services," *ACM Transactions on Computer Systems*, Vol. 16 No. 4, November 1998, pp. 321-366.
3. Birman, K. "The Process Group Approach to Reliable Distributed Computing," *Communications of the ACM*, Vol. 36 No. 12, December 1993, pp. 37-53 and 103.
4. Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service", *ACM Transactions on Computer Systems*, 19(3):332-383, Aug 2001.
5. Devanbu, P., M. Gertz, C. Martel, and S. Stubblebine, "Authentic Third-Party Data Publication", in Proceedings of the Fourteenth IFIP Working Conference on Database Security, August 2000.
6. Elder, M.C. "Fault Tolerance in Critical Information Systems," Ph.D. dissertation, Department of Computer Science, University of Virginia, May 2001.
7. Hall, R.M., "Agent-based Software Configuration and Deployment," University of Colorado Ph.D. Thesis, April 1, 1999.
8. Hall, R.M., D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. "An Architecture for Post-Development Configuration Management in a Wide-Area Network" in Proceedings of the 1997 International Conference on Distributed Computing Systems, pages 269-278. IEEE Computer Society, May 1997.
9. Hall, R.M., D.M. Heimbigner, and A.L. Wolf., "Evaluating Software Deployment Languages and Schema", in Proceedings of the 1998 International Conference on Software Maintenance, pages 177- 185. IEEE Computer Society, November 1998.
10. Hall, R.M., D.M. Heimbigner, and A.L. Wolf. "A Cooperative Approach to Support Software Deployment Using the Software Dock", in Proceedings of the 1999 International Conference on Software Engineering, pages 174-183. Association for Computer Machinery, May 1999.
11. Hofmeister, C., E. White, and J. Purtilo. "Surgeon: A Packager for Dynamically Reconfigurable Distributed Applications," *Software Engineering Journal*, Vol. 8 No. 2, March 1993, pp. 95-101.
12. Knight, J.C., M. C. Elder, "Fault Tolerant Distributed Information Systems", International Symposium on Software Reliability Engineering, Hong Kong (November 2001).
13. Knight, J., M. Elder, J. Flinn, and P. Marx. "Summaries of Three Critical Infrastructure Systems," Technical Report CS-97-27, Department of Computer Science, University of Virginia, November 1997.

14. Knight J.C. and K.J. Sullivan, "On the Definition of Survivability", University of Virginia, Department of Computer Science, Technical Report CS-TR-33-00.
15. Magee, J., N. Dulay, S. Eisenbach, and J. Kramer. "Specifying Distributed Software Architectures," *Lecture Notes in Computer Science*, Vol. 989, September 1995, pp. 137-153.
16. Magee, J. and J. Kramer. "Darwin: An Architectural Description Language," <http://www-dse.doc.ic.ac.uk/research/darwin/darwin.html>, 1998.
17. Office of the Undersecretary of Defense for Acquisition and Technology. "Report of the Defense Science Board Task Force on Information Warfare - Defense (IW-D)," November 1996.
18. Porras, P.A. and P. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances", Proceedings: 20th National Information Systems Security Conference, October 1997.
19. President's Commission on Critical Infrastructure Protection. "Critical Foundations: Protecting America's Infrastructures The Report of the President's Commission on Critical Infrastructure Protection," United States Government Printing Office (GPO), No. 040-000-00699-1, October 1997.
20. Sullivan, K., J. Knight, X. Du, and S. Geist. "Information Survivability Control Systems," Proceedings of the 21st International Conference on Software Engineering, IEEE Computer Society Press, Los Alamitos, CA, May 1999, pp. 184-192.
21. Summers, B. *The Payment System: Design, Management, and Supervision*, International Monetary Fund, Washington, DC, 1994.
22. United States Air Force, Joint Battlespace Infosphere Home Page, <http://www.rl.af.mil/programs/jbi/default.cfm>
23. Wang, C., A. Carzaniga, D. Evans, and A.L. Wolf, "Security Issues and Requirements for Internet-scale Publish-Subscribe Systems", in Proceedings of the Thirtyfifth Hawaii International Conference on System Sciences (HICSS-35), Big Island, Hawaii, Jan. 2002.
24. Wang, C., J. Davidson, J. Hill, J. Knight, "Protection of Software-based Survivability Mechanisms", International Conference of Dependable Systems and Networks, Goteborg, Sweden (July, 2001)

3. SUMMARY OF RESEARCH FINDINGS

Recall that the goal of the project was to assess the utility of the various research activities in the Willow project to the JBI concept. In this section, we summarize the technical results of the research project. Note that these summaries are intended to provide only a brief overview since the detailed results are contained in the various papers and results detailed in section 5.

3.1. Willow Architecture And JBI Survivability

Our first finding is that *the Willow architecture makes the JBI survivable*. The JBI concept is vulnerable to the entire set of fault classes that the Willow architecture is designed to handle. For example, a JBI will be subject to improper deployment, erroneous configurations, random hardware component failures, software component failures, mistakes by operators, unpredictable battle damage, and a variety of security attacks. The application of all of the basic Willow architectural ideas to the JBI concepts has been investigated. This investigation has been both theoretical and experimental. From a theoretical viewpoint, it has been shown that all of the fault classes that the Willow architecture is designed to treat are handled effectively for the JBI. As a practical demonstration, two prototype systems, System 1 and System 2, were built. System 1 consisted of a preliminary implementation of the Willow architecture and a small example JBI system. This system included a mechanism for injection of faults. The deployment of the JBI application and the control of the Willow system were automatic and faults injected into the JBI system were shown to be tolerated. System 2 was far more extensive and designed to be a scalable Willow implementation that includes all the necessary facilities to support a production-sized JBI implementation. This second prototype has been used to both demonstrate capabilities and to perform some initial performance measurements. Both prototype systems used experimental JBI implementations that were based on the Siena publish/subscribe system, and both included realistic JBI services. The second prototype JBI application featured support for an arbitrary number of clients with flexible subscription and publication services, multiple data sources including equipment position updates and simulated weather information, and a set of supplementary data repositories. The mode of operation for this latter prototype was to exploit the publish/subscribe infrastructure for both notification and moderate-sized data items and to use the supplementary data repositories for large data volumes such as air-tasking orders and maps.

3.2. Probabilistic Risk Analysis And JBI Security

Our second finding is that *hazard analysis and fault-tree analysis facilitate reasoning about JBI security*. An analysis was undertaken that applied the techniques of hazard and fault-tree analysis to the problem of reasoning about security of complex networked information systems. The goal was to determine whether such techniques are helpful in the detection of hazards and the identification of possible attacks. The basic event concept that is used in the construction of fault trees for safety cases was extended for security by separating events that lead to vulnerabilities from events associated with attacks. The fault tree that was developed was based upon minimal assumptions about the form of a typical JBI so that the results would be generally useful. The conclusion of the activity was that the systematic development of fault trees for a JBI based on hazards that reflect security concerns produces a systematic and comprehensive (though necessarily informal) tabulation of the JBI's threats and vulnerabilities.

3.3. Publish/subscribe Security

Our third finding is that *publish/subscribe communication induces both traditional and novel security concerns*. Authentication of data sources and integrity of the data can be guaranteed using traditional signature-based mechanisms. Similarly, authentication of dissemination nodes and clients, and privacy of communication between pairs of dissemination nodes, and between clients and dissemination nodes can be guaranteed using traditional mechanisms, such as SSL connections. However, these schemes assume that dissemination nodes are permitted access to subscriptions and to messages. If they cannot be trusted to see either subscriptions or messages, then a completely new set of techniques will be required to secure them. The reason is that the subscriptions are used to route the messages. No techniques are known for routing messages based on subscriptions when either or both are not visible to the router.

3.4. Non-Military JBI Applicability

Our fourth finding is that *the JBI architectural concepts are useful beyond military applications*. In a separate project we have built a demonstration application based on a JBI-like architecture for helping forest-fire fighters to manage information. The application takes satellite images (visual and infrared), topographic information, and weather information, fuses them, and then presents them to low-powered devices (handheld computers). In addition, the low-powered devices, which are what fire-fighters in the field would use, can annotate the information (e.g., marking the fire boundary) and publish that information back out to other fire fighters and the incident commander. This application was built using the Siena wide-area publish/subscribe communication service.

4. PERSONNEL ASSOCIATED WITH RESEARCH EFFORT

During the period of performance of this research effort the following faculty and professional staff were associated with the project:

Name	-	John C. Knight
Position	-	Professor
Institution	-	Dept. of Computer Science, University of Virginia
Name	-	Alexander Wolf
Position	-	Professor
Institution	-	Dept. of Computer Science, University of Colorado
Name	-	Dennis Heimbigner
Position	-	Research Associate Professor
Institution	-	Dept. of Computer Science, University of Virginia
Name	-	Antonio Carzaniga
Position	-	Research Assistant Professor
Institution	-	Dept. of Computer Science, University of Colorado

The following graduate students were associated with this project:

Name	-	Matthew Elder
Advisor	-	John C. Knight
Thesis/Dissert. Title	-	Fault Tolerance in Critical Infrastructure Systems
Degree	-	Ph.D.
Status	-	Graduated May, 2001
Name	-	Chenxi Wang
Advisor	-	John C. Knight
Thesis/Dissert. Title	-	A Security Architecture For Survivability Mechanisms
Degree	-	Ph.D.
Status	-	Graduated May, 2001
Name	-	Jonathan Hill
Advisor	-	John C. Knight
Thesis/Dissert. Title	-	To be determined
Degree	-	Ph.D.
Status	-	In progress, expected May, 2003
Name	-	Michael Tashbook
Advisor	-	John C. Knight
Thesis/Dissert. Title	-	To be determined
Degree	-	Ph.D.
Status	-	In progress, expected May, 2003
Name	-	Philip Varner

Advisor	-	John C. Knight
Thesis/Dissert. Title	-	Formal Specification of Survivability
Degree	-	M.S.
Status	-	In progress, expected May 2003
Name	-	Matthew Rutherford
Advisor	-	Alexander L. Wolf
Thesis/Dissert. Title	-	EJB-ARK: Enterprise JavaBean Automatic Reconfiguration frameworkK
Degree	-	M.S.
Status	-	Graduated December 2001
Name	-	Christopher Corliss
Advisor	-	Alexander L. Wolf
Thesis/Dissert. Title	-	Using the Structure of XML to Generate Notifications and Subscriptions in Siena
Degree	-	M.S.
Status	-	Graduated May 2002
Name	-	Nathan Ryan
Advisor	-	Alexander L. Wolf
Thesis/Dissert. Title	-	TBD
Degree	-	Ph.D.
Status	-	In progress

The following undergraduate students were associated with this project: G.J. Halfond.

5. PUBLICATIONS

During the period of this research project, the papers contained in the following list were prepared. Copies have been provided under separate cover:

1. J. Knight and K. Sullivan, "On the Definition of Survivability", University of Virginia, Department of Computer Science Technical Report TR-00-33, December 2000.
2. D. Heimbigner, "Adapting Publish/Subscribe Middleware to Achieve Gnutella-like Functionality", 2001 ACM Symposium on Applied Computing (SAC 2001) pp. 176--181, Las Vegas, NV, March 2001.
3. C. Wang, J. Davidson, J. Hill, J. Knight. "Protection of Software-based Survivability Schemes". IEEE/IFIP International Conference of Dependable Systems and Networks. July 2001.
4. A. Carzaniga, D.S. Rosenblum, and A.L. Wolf. "Design and Evaluation of a Wide-Area Event Notification Service", ACM Transactions on Computer Systems, Vol. 9, No. 3, August 2001.
5. J.C. Knight and M.C. Elder, "Fault Tolerant Distributed Information Systems", International Symposium on Software Reliability Engineering, Hong Kong, November 2001.
6. A. Carzaniga, J. Deng, and A.L. Wolf, "Fast Forwarding for Content-Based Networking", Technical Report CU-CS-922-01, Department of Computer Science, University of Colorado, Boulder, Colorado, November 2001.
7. J.C. Knight, D. Heimbigner, A.L. Wolf, A. Carzaniga, J. Hill, P. Devanbu, and M. Gertz, "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications", Technical Report CU-CS-926-01, Department of Computer Science, University of Colorado, Boulder, Colorado, December 2001.
8. C. Wang, A. Carzaniga, D. Evans, and A.L. Wolf. "Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems", Hawaii International Conference on Systems Science, January 2002.
9. P. Varner and J.C. Knight. "Security Monitoring, Visualization, and System Survivability", Fourth Information Survivability Workshop, March 2002.
10. J.C. Knight, D. Heimbigner, A.L. Wolf, A. Carzaniga, J. Hill, P. Devanbu, and M. Gertz. "The Willow Survivability Architecture", Fourth Information Survivability Workshop, March 2002.
11. A. Carzaniga, J. Deng, and A.L. Wolf, "A Benchmark Suite for Distributed Publish/Subscribe Systems", Technical Report CU-CS-927-02, Department of Computer Science, University of Colorado, Boulder, Colorado, April 2002.
12. M.J. Rutherford, K. Anderson, A. Carzaniga, D. Heimbigner, and A.L. Wolf, "Reconfiguration in the Enterprise JavaBean Component Model", IFIP/ACM Working Conference on Component Deployment, Lecture Notes in Computer Science 2370,

Springer-Verlag, Berlin, June 2002.

The following reports were prepared during the period of the project and are included in this report as appendices:

1. C. Wang, A. Carzaniga, D. Evans, and A.L. Wolf. “Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems”.
2. R. Schutt. “Domain Analysis of the Joint Battlespace Infosphere”.
3. J. Knight and R. Schutt. “The Application of Probabilistic Risk Analysis to Security”.

6. PRESENTATIONS GIVEN

As a result of this research effort, presentations that included discussion of various elements of the research have been given at a variety of organizations over and above principal-investigator meetings and professional conferences. The presentations given were:

Organization	-	CERT/SEI
Date	-	January 18, 2001
Presentation Title	-	Willow: A Comprehensive Approach to Survivability
Organization	-	Michigan State University
Date	-	January 30, 2001
Presentation Title	-	Survivability of Critical Infrastructure Systems
Organization	-	North Carolina State University
Date	-	March 6, 2001
Presentation Title	-	Survivability of Critical Infrastructure Systems
Organization	-	University of Colorado
Date	-	November 15, 2001
Presentation Title	-	Bend, Don't Break: Using Reconfiguration to Achieve Survivability
Organization	-	Georgia Institute of Technology
Date	-	December 5, 2001
Presentation Title	-	Bend, Don't Break: Using Reconfiguration to Achieve Survivability
Organization	-	National Academy of Sciences, Committee on Terrorism
Date	-	February 25, 2002
Presentation Title	-	Terrorism & Critical Infrastructure Sector Vulnerability: A Software Engineering Systems Perspective
Organization	-	U.S. Air Force, Rome Laboratories
Date	-	July 9, 2002
Presentation Title	-	Critical Infrastructure Protection: A Software Systems Perspective
Organization	-	Central Intelligence Agency Terrorism Workshop
Date	-	July 10, 2002
Presentation Title	-	Software Engineering: The Root Cause?

7. DISCOVERIES, INVENTIONS, AND PATENTS

All of the results achieved in this research have been or will be documented in papers in the archival literature. No patents have been applied for and none are expected.

APPENDIX A

“Domain Analysis of the Joint Battlespace Infosphere”

DOMAIN DESCRIPTION OF THE JOINT BATTLESPACE INFOSPHERE

DOMAIN DESCRIPTION DOCUMENT AND REQUIREMENTS

Robert Schutt

rschutt@cs.virginia.edu



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF VIRGINIA

151 ENGINEER'S WAY
CHARLOTTESVILLE, VIRGINIA 220904
(804) 982-2220

Table of Contents

1 Document Scope	32
2 The JBI	32
3 JBI Domain Definition	33
4 JBI Requirements	34
4.1 High Level Requirements for a JBI Implementation.....	34
4.2 Functional Requirements	34
4.2.1 Information Content Requirements.....	34
4.2.2 Data Class Requirements.....	35
4.2.3 Information Brokering Requirements	35
4.2.4 Aggregation / Interpretation Requirements.....	35
4.2.5 Publication Mechanism	36
4.2.6 Query mechanism	36
4.2.7 Subscription mechanism	36
4.2.8 Hardware / Platform Requirements	37
4.2.9 Human / JBI Interface Requirements.....	37
4.3 Non-functional Requirements	38
4.3.1 Quality of service requirements.....	38
4.3.2 Security Requirements	38
4.3.3 Information Consistency	39
4.3.4 Future Integration Requirement.....	39
4.3.5 Rapid deployment requirement.....	39
4.3.6 Critical task automation requirement	39
4.3.7 Management and control functions.....	39
4.3.8 Small forward footprint.....	40
4.3.9 Self-monitoring	40
4.3.10 Reliability Requirements.....	40
5 The JBI Solution	40
5.1 Technical Background	40
5.1.1 Event Notification (Publish / Subscribe).....	41
5.1.2 Software Agents.....	41
5.2 Information exchange through “publish / subscribe”	42
5.3 Transforming data into knowledge via fuselets	43
5.4 Distributed collaboration through shared, updateable knowledge objects	44
5.5 Assigned unit incorporation via force templates	44
6 Conclusions.....	45
JBI Dictionary.....	47
References	49

1 Document Scope

The purpose of this document is to analyze the “domain” of the JBI to provide a technical basis for understanding JBI technologies and requirements. To accomplish this analysis, the following process followed:

Define the JBI Domain – Important “customers” are identified and the operating environment for the JBI is discussed.

Model the JBI Domain – The deployment and functionality for the JBI is described.

Identify Requirements – The high-level JBI requirements are identified to understand the implementation constraints.

The JBI Solution – The proposed architecture suggested by the USAF is discussed.

The references for this document are published technical reports, slides, websites, and unpublished draft documents from organizations working on JBI technologies. It is necessary to include information contained in unpublished “working documents” to provide the most up to date analysis of the JBI. If the references are unavailable they may be obtained by contacting rschutt@cs.virginia.edu.

2 The JBI

The JBI was proposed in a December 1999 report by the United States Air Force Scientific Advisory Board. This report outlined the JBI vision and began the process of turning the JBI from an idea into a reality. Since 1999 government contractors and the scientists in the USAF have been working on the JBI. The timeline for the JBI has several milestones but the USAF predicts completion of the final work and an operational JBI by 2010 [12, 144].

What is the US-SAB vision for the JBI? It is perhaps best summarized by the following two quotes:

“The primary goal of the JBI is to provide the right information, to the right person, at the right time, using the right media, language, and level of detail during the planning, command, execution and combat support of missions.” [12]

“full, real-time battlespace awareness to users at all echelons” [12, 2]

The Air Force JBI solution is built on four enabling technologies [12, iv]:

Information Exchange through “publish and subscribe” – Publish / Subscribe is an information distribution mechanism based on the concept of “content based routing.” In

content based routing the destination of a message is based on the content of the message rather than a receiver address.

Transforming data into knowledge via fuselets. – Fuselets are autonomous agents that interact with the publish / subscribe mechanism transforming the flood of available information into knowledge useful for decision making.

Distributed collaboration through shared, updateable knowledge objects – To enable distributed collaboration, it is necessary that all users have a consistent view of the battlespace. Coupled with the consistent view of the battlespace is an ontology. This ontology is a pre-specified “dictionary” of allowable objects and knowledge about the relationships between objects.

Assigned unit incorporation via force templates. - To allow units to quickly enter the battlespace, information requirements are pre-specified in “force templates.” These force templates are sets of publish and subscribe requests that allow the unit to quickly plug into the information infrastructure.

There is no single JBI. JBI is a set of software solutions that are “JBI compliant” from which a JBI can be built. Many authors have stated that it is not correct to speak about “the JBI” because this implies a single implementation [8]. It is more correct to speak about “a JBI” which means one particular implementation of JBI. The specification for JBI compliant software has not been established at the writing of this document.

3 JBI Domain Definition

To understand the JBI service interface important customers of the JBI are discussed. Each customer can interact with the JBI in two important capacities. First, customers can contribute to knowledge of the battlespace by submitting information that will be integrated into the JBI. Secondly, customers can learn from other participants by requesting information from the JBI.

Commander – The commander is the top level consumer of JBI information. At the command level, the JBI provides accurate and concise statements about the state of the battlespace. These statements are aggregated assertions that are a result of the mass of information flowing in from lower-level users. This aggregated information aids the commander in making better decisions.

Field Units – Field units implement the actions requested by the commander and collect information about the battlespace to share with other users. Examples of field units are tanks, infantry, satellites and aircraft. Field units have both information requirements and information capabilities. Information requirements may be orders issued by the commander or locations of enemy units. Information capabilities might be unit status information like ammunition, fuel, and health. In the case of reconnaissance units the unit may be capable of publishing information about enemy locations.

Support Units –Support units provide for the maintenance and replenishment of fighting units. There is little difference in the service interface provided by the JBI to support units.

IT Support Staff – The IT Support Staff serves in the critical role of administering the JBI. There are three stages in the JBI lifecycle: “standing-up”, operation, and “standing-down”. “Standing-up” a JBI is deploying it to serve the operation purpose of the force. “Operation” is maintaining the JBI in operational form even in the presence of battle damage. During operation the IT Support Staff may be involved investigating attacks against the IT infrastructure or reconfiguring in response to damage or anticipated threats. In addition, the IT staff is responsible for more mundane administration tasks like maintaining the authorization/access policy. “Standing-down” the JBI is terminating the JBI at the end of the operation. This may involve archiving information for later use or investigation.

Analysts – Analysts look at low level information in the battlespace and make inferences about higher level knowledge. In the JBI analysts may be either computer or human. As an example an analyst may look enemy movement information and postulate that the enemy is making a new offensive maneuver. Analysts are critical to the JBI because they add the “intelligence” to the system by making important inferences.

4 JBI Requirements

It is first important to consider the requirements for a JBI implementation before considering the technical solution to the problem of battlefield information management. This section summarizes the requested end-to-end capabilities of a JBI implementation quoting directly from USAF documents. The following requirements are completely separate from any implementation direction as to not limit the design space of the JBI.

4.1 High Level Requirements for a JBI Implementation

JBI requirements are broken down into two categories: functional requirements and non-functional requirements.

4.2 Functional Requirements

4.2.1 Information Content Requirements

“[a JBI] provides individual users with specific information required for their functional responsibilities” [12, iii]

A JBI manages information that is related specifically to the users in the system. The information managed by the JBI will be related to the functional tasks of the users. The JBI is to act in a “support role” to users, providing the information needed to accomplish their tasks in the combat environment.

4.2.2 Data Class Requirements

“... commander’s level, the JBI is a powerful command and control system that combines inputs from a variety of sources, including existing C² systems, reconnaissance data, satellite data, unit capability data, logistics data, and real-time battlefield conditions.” [12, iii]

The JBI must manage data with a variety of properties:

- 1) Pre-specified data: data that is stored prior to deployment of JBI. Example: ‘unit capability’
- 2) Low-volume real-time data: information that has moderate to low bandwidth requirements. Example: ‘logistics data’
- 3) High-volume real-time data: data with high bandwidth or ‘streaming’ requirements. Example: ‘satellite data.’

4.2.3 Information Brokering Requirements

The JBI will be a bridge between existing systems, similar to information sharing in a business to business (B2B) relationship in the commercial sector.

Information Translation

“The JBI platform integrates many individual information systems that currently support operational forces. Each existing system has been developed in a stovepiped fashion; few interoperate with each other. The JBI acts as an intermediary between these systems, converting information from one representation to another to enable interoperability.” [12, iii]

The JBI must function as the link between information systems, transforming information produced by one system into a format that can be consumed by another system.

Backward Compatible Integration

“the JBI is a powerful command and control system that combines inputs from a variety of sources, including existing command and control systems.” [12, iii]

The JBI must broker information between existing command and control systems.

4.2.4 Aggregation / Interpretation Requirements

Aggregation of information

“the JBI builds an aggregated picture from these combined inputs [satellite data, etc].” [12, iii]

Aggregation during brokering

“the JBI interprets the information flowing between applications, using it to build its own, more complete, picture of the current situation.” [12, iii]

During the process of brokering information between applications (some systems will be pre-existing systems that are not ‘JBI aware’), the JBI must extract information from these sources and aggregate this information.

Aggregation pedigree

“The JBI maintains pedigree information as it aggregates information, so the commander can drill down to examine the inputs and processes the JBI uses to generate an aggregated piece of higher-level knowledge.”

Modification of aggregation

“the JFC information management staff ... can write scripts or aggregate information in a new way.” [12, v]

A runtime mechanism must exist for modifying how information is aggregated in a JBI.

4.2.5 Publication Mechanism

Publishing is a generic term for ‘adding’ information to the system. A unit may ‘publish’ their location or the location of an enemy base. A publication can also ‘update’ a data in a previous publication.

“JBI data can be published” [12, 1]

4.2.6 Query mechanism

“JBI information can be searched to meet user needs.”

4.2.7 Subscription mechanism

“JBI data can be ... subscribed to” [12, 1]

A subscription is a query for future data. An example is that a soldier may subscribe to be notified when an enemy is within 10 miles of their location.

4.2.8 Hardware / Platform Requirements

Global Information Grid Requirement

“the JBI provides services through a federation of multiple servers, the Global Information Grid connects these servers to each other and to the many systems that support the JBI” [12, iii]

The transport mechanism for information will be the Global Information Grid.

Mobile / Unique Device Requirement

“Information is delivered by an appropriate interface: on a soldier’s handheld computer, the pilot’s head-up display, or the virtual-reality collaborative environment within which many users share information and explore alternative courses of action through simulation.” [12, v]

The system must interface with a wide variety of hardware devices.

4.2.9 Human / JBI Interface Requirements

Situational Awareness Requirement

“The JBI builds an aggregated picture from these combined inputs [satellite data, etc], giving unparalleled situational awareness accessed as easily as a web page.” [12, iii]

The JBI must present information to provide a JBI user with “situational awareness.” This requirement presents a significant problem for an implementation of the JBI. The quantity of information available is meaningless unless there is a mechanism for synthesizing this information into a form that can provide “human awareness.”

Individual View Requirement

“... the JBI tailors this picture [the view of the current situation] for individual users: the commander gets a high-level view of the campaign, while the soldier in the field gets a detailed description of a nearby hostile base.” [12, iii]

The presentation of information is dependent on the particular user of the system.

JBI Browser

“Some users will want to browse information in the JBI, much the way they browse the Web today. The browser will be able to fetch objects from the JBI, search it using

queries, and to make visual presentations of many of the common JBI information objects.” [12, 82]

4.3 Non-functional Requirements

4.3.1 *Quality of service requirements*

‘Downward flow’ requirement

“The JBI provides for speedy downward flow of information, so when commanders order an action, the action is received and implemented at the subordinate level almost immediately.” [12, iii]

The JBI will be used to issue commands to subordinates. The latency between the time the order is issued and the time the action is received is to be “almost immediate.” The vagueness the term “almost immediately” introduces is troublesome for a JBI implementation.

Bandwidth throttling

“the JBI monitors the available communication bandwidth. If some of the data links become degraded or lost, the JBI reduces information flow so that no links become saturated.” [12, 27]

Levels of Service

“the JBI platform may be configured to allocate different classes of service to different clients: some may require fast response, some may have low priority, some may not be permitted to access all objects in the JBI, and so forth.” [12, 82]

4.3.2 *Security Requirements*

“[the JBI must] limit access to critical information, ...” [12, iii]

Access Controls

“the JFC’s information management staff ... can change access controls on information”

Authentication

“Identification and authentication are key to the ability to ensure that the support products and user-generated products come from sources that are properly labeled to indicate information pedigree. Similarly identification

and authentication are used to verify that commands and execution activities are entered by legitimate users and that the interactions in the JBI are conducted by people with the appropriate authority.” [12, 7]

“... to verify the identity of JBI clients and JBI users, the JBI must authenticate them ...” [12, 91]

4.3.3 Information Consistency

“... the information delivered is timely and consistent with the information shared with all JBI users.”

4.3.4 Future Integration Requirement

“The JBI provides an architecture for the incorporation of future data capture technologies that exploit better sensors, databases, fusion engines, automated analysis tools, collaborative planning and execution aids, and distribution controls.” [12, iv]

4.3.5 Rapid deployment requirement

“The commander in chief (CINC) or JTF commander creates a JBI for a specific purpose, usually in response to a crisis or conflict. The JBI enables the commander to focus information support for a specific operational purpose, ...”

“The JBI supports [the USAF being an expeditionary force] by being easily and quickly deployed.” [12, 2]

A JBI implementation will be deployed on demand and customized to support an “operational purpose.” Because the JBI will be deployed in response to a crisis where response time is often critical, this requirement suggest the deployment process should be rapid.

4.3.6 Critical task automation requirement

“The JBI eases system management through automation of critical tasks.”

4.3.7 Management and control functions

“The JBI must include management and control functions. These tools monitor and control such aspects as JBI establishment, performance, bandwidth allocation, security, data management, network configuration, and repair or restoration.” [12, 8]

4.3.8 Small forward footprint

“The JBI has a small forward footprint, and it provides many of its services via reachback connections. Much of its computational power and storage capacity reside in the rear., with a minimum number of components near the battlefield.” [12, 27]

4.3.9 Self-monitoring

“the software will need to monitor its own operation and report “alarms” to JBI information management staff.”

Forensics and damage assessment

“Analyze what information, system functions, and decisions have been affected by adversary activities, When a new or unknown type of attack is suspected, examine system state, software and support data, to determine its means of operation and identify ways of removing, isolating or otherwise countering it.” [12, 121]

4.3.10 Reliability Requirements

24 * 7 operability

*“The JBI must be designed to operate in a 24 * 7 mission-critical environment” [12, 92]*

Critical information availability

“critical information for a mission must be available when needed, in the presence of failures and attacks.” [12, 93]

5 The JBI Solution

The solution proposed by the USAF to meet the requirements of a JBI is based on the four architectural concepts [12, iv]:

1. Information Exchange through “publish and subscribe”.
2. Transforming data into knowledge via fuselets.
3. Distributed collaboration through shared, updateable knowledge objects
4. Assigned unit incorporation via force templates.

These four architectural principles are the foundation of the JBI architecture.

5.1 Technical Background

To better understand the JBI a brief overview of central JBI technical concepts is presented.

5.1.1 Event Notification (Publish / Subscribe)

Event notification is a messaging paradigm based on the principle of “content based routing.” In content based routing, the routing of information is based on the content of the message rather than the address of the receiver. Event notification systems are also called “publish / subscribe” systems because of the application programming interface (API) used by clients. To receive information from a event notification system a client “subscribes” to be notified about future events. In order to subscribe to an event, the subscriber must provide a subscription criteria. The format for subscription criteria varies widely among publish / subscribe systems but most use some “SQL-like” interface. An example of a subscription might be:

“SUBSCRIBE WHERE (ticket_type = ‘airline_ticket’) AND (destination = ‘Chicago’) AND (origin = ‘Charlottesville’) AND (price < \$400)”

Events are generated by “publishers.” To create a new event the publisher binds “attributes” with “values.” The attributes in the above example are: ticket_type, destination, origin, and price. The values in the above example are: airline_ticket, Chicago, Charlottesville, and \$400. Once the publisher has bound values to attributes, the event is then submitted to the publish / subscribe system to be routed to clients.

It is important to note that event notification is a more general paradigm than traditional routing. To do traditional routing with a publish / subscribe system, each client subscribes to be notified about all messages where “address = [my IP address]”. Each publisher specifies the receiver address in the message.

5.1.2 Software Agents

A “software agent” is program that helps users accomplish tasks in a system. The agent research community has defined several desirable properties for software agents [5]:

Autonomous – They do not require human interaction.

Goal Oriented – Function to accomplish a goal in the system.

Proactive – capable of taking initiative, not only “reacting” to the world around them.

Responsive – the program learns user preferences

Adaptive – the agent modifies behavior to exist in a changing environment

It is obvious that the above goals are unrealistic given the current state of Computer Science. “Autonomous” and “Goal Oriented” are the most relevant agent properties for understanding how agents will be used in a current JBI implementation.

There are two types of software agents: mobile agents and static agents. Mobile agents can move from one system to another while accomplishing their tasks. Static agents

reside on one system throughout their lifetime. The difference between mobile agents and static agents is not in their function, but rather in the complexity that is introduced by allowing executable code to move between systems.

5.2 Information exchange through “publish / subscribe”

The “JBI platform” is the name for the “publish / subscribe” middleware in the JBI [12, 89]. The “publish / subscribe” middleware envisioned for use in a JBI is beyond the capabilities of today’s state of the art. There are two main reasons why current publish / subscribe software cannot be directly used in a JBI implementation. First, the fact that a JBI operates in a hostile environment where extensive damage is the norm rather than the extreme imposes certain requirements not dealt with by today’s publish / subscribe systems. Secondly, documents by Mitre have suggested that JBI must include filtering capabilities not available in state of the art publish / subscribe systems.

The USAF states that the “publish / subscribe” component of the JBI must provide the following services [12, 89]:

Publish: Clients must be able to publish new attribute-value pairs. When a client publishes information it is routed to other clients in the system. When the value of an attribute changes the client must “republish” the information to notify subscribers of the new value. Clients can “delete” previously published objects from the system. The effect of deleting an object is that it no longer available to other clients in the system.

Query: Clients may search the JBI for patters that match the attribute/value pairs. To query the JBI a client submits a pattern to the JBI. Objects with attribute/value that match the pattern are sent to the client. An important aspect of the query mechanism in JBI is that USAF has requested that a set of “common searchable attributes” be built into the JBI. Searches on a common searchable attribute will be executed faster then on a general searchable attribute [12, 90]. Common searchable attributes include type of object, the object identifier, the creator, and the geospatial-temporal region associated with the object.

Subscribe: Subscribe is a query for future information. When an object is published that matches a clients pattern a “trigger” executes and the client is notified [12,90]. The attribute value pairs that trigger the notification are returned to the client [12, 90].

Transform: The services for running fuselets (programs that publish and subscribe).

Control: Services to control the configuration of the JBI. Software will be used by the information staff managing the running JBI to accomplish the configuration control.

Access Control: The JBI must be able to authenticate and authorize access to information disseminated by the JBI.

Scott Renner in his slides “Explaining the JBI Reference Architecture: A Series of Lies” presents some of the common misconceptions about the JBI. This document is important because it indirectly suggests a set of functional requirements for the JBI publish / subscribe middleware.

Most publish / subscribe systems place the burden of routing and filtering messages on the internal router nodes of the system. In Renner’s document these are referred to as “dissemination nodes.” To handle the information requirements of a JBI, Renner suggests that filtering and routing must occur at the client, the router, and the subscriber [8]. Client side filtering is invaluable because in the JBI there are likely to be many repeated broadcast messages. An example of a repeated broadcast message is a unit that updates its position every 10 seconds. Without client side filtering the message will be submitted to the router nodes whether or not there are subscribers interested in the unit’s position. If a client can filter and route this information appropriately without involving the internal router nodes, it may be possible to prevent these “heart beat” type messages from flooding the information network. Conversely, there may be large streams of information which may be too costly to filter at the client or at router nodes. Examples of this may be satellite or mapping data. In this instance, the routing system can provide a “superset” of the requested information directly to the client. To complete the query the client processes the information using its own CPU capacity.

5.3 Transforming data into knowledge via fuselets

Fuselets are software agents in the JBI that have the operational goal of fusing information into knowledge. Knowledge to the USAF, means “decision-quality information” [2], that is, information that can be directly used to make decisions. Decision-quality information needs no further analysis or interpretation. As an example, suppose a college admits students who have a GPA higher than 3.0. If a student presents the admissions committee with a list of classes and corresponding grades, eg CS 101: A, CS 102: B, ..., the admission committee has information to make a decision, but the information is not yet “decision-quality.” After the GPA has been computed, the admissions committee now has “decision-quality information” because they can either accept or reject the student by directly using the information.

In the near future, fuselets will be scripts that interact with the JBI Platform publishing and subscribing to information. In a near-term JBI implementation, fuselets will be neither mobile nor intelligent [2]. The USAF writes that fuselets will function in the following capacities [12, 69]:

Search – Fuselets will search the JBI for information. As an example, the USAF suggests a fuselet could be used to query sources for a supplier who could provide equipment necessary for a logistic function [12, 69].

Collate – Fuselets will compare information from multiple sources, examining the data to note points of disagreement.

Identify – Many of the information objects in the combat environment are higher level objects that are often not detected directly. In this instance, *aggregation fuselets* can be used to identify higher level objects by looking at lower level information. A “tank group level” fuselet may look at a several tanks (lower level objects) and suggest the creation of a tank group [12, 73].

Update – As many of the information objects in a JBI will be high level objects, fuselets will need to update these objects when low level information changes.

Monitor – Fuselets will be used monitor the stream of information flowing through a JBI to pick out pertinent information.

To augment the synthesis of information in the JBI the Air Force suggests the use of “human analysts” to accomplish the tasks that “computer only” fuselets cannot provide [12, 80]. These analysts will interact with JBI just like fuselets, subscribing to information and publishing new information to the JBI. The only difference between a fuselet and a human analyst is that the processing of information is done by a human instead of a computer. Special software must be created to allow analysts to interact with JBI.

5.4 Distributed collaboration through shared, updateable knowledge objects

The JBI allows users to collaborate to make decisions. To facilitate this collaboration the JBI keeps a set of objects that represent the “common operating picture.” [12, vii]. This common operating picture is called the “Structured Common Representation” and abbreviated as the SCR. The SCR is structured because it is both hierarchical and composed of sets or collections [12, 66]. The upper levels of the hierarchy have information that is useful at the command level [12, 66]. The hierarchy is ordered by the complexity of the object, for example, a battalion is above tank group and tank group is above an individual tank.

In order to have a meaningful SCR, the USAF suggests a set “ontology” or dictionary of allowable objects in the SCR. [12, 66]

“The ontology is the JBI’s vocabulary. If there is not an ontology entry, the object or concept cannot enter the JBI.” [12, 66]

The SCR is the input for the many different collaborative mechanisms envisioned by the USAF. There will likely be a wide variety of mechanisms for interacting with the SCR including computer visualization, virtual reality, collaborative whiteboards, etc. These end user devices are documented extensively in the USAF SAB Report “Building the Joint Battlespace Infosphere Volume 2: Interactive Technologies.” [13]

5.5 Assigned unit incorporation via force templates

A force template is the software description of a military unit that can participate in the JBI [12, vii]. To enter a JBI the unit specifies its information requirements and capabilities in a force template. As an example an infantry soldier may subscribe to be notified about all enemy units within 1 mile of their position. This subscription request would be contained in a force template and integrated into the JBI when the unit enters the battlespace.

A force template is the key enabling technology that makes a JBI rapidly deployable. A new unit with a force template can quickly enter the battlespace because the information requirements have been determined and its position in the ontology established.

There are two main types of force templates that correspond to two the two main types of units in a battlespace, support unit force templates and fighting unit force templates. The Air Force suggests fighting unit templates will contain the following fields: force employment capability, ammunition inventory, communication requirements, computing systems, information requirements (subscriptions for the unit's clients), information products (objects to publish), and personnel requirements [12, 33]. No further elaboration of the contents of these fields is available from the USAF.

The Air Force states that all support force templates will contain the following four information fields [12, 34]:

Information requirements – the basic subscriptions for the unit.

Information products – the objects the unit will publish to the JBI.

Communication Requirements – No information about the contents of this part of the force template is available from the Air Force.

Computing Systems – No information about the contents of this part of the force template is available from the Air Force.

The USAF states that support unit force templates may contain other fields, but all support force templates will contain at least the above mentioned four fields. [12,34] The fact that the fighting unit force template also contains the fields required for the support unit force template suggests that the four fields (information requirements, information products, communication requirements, computing systems) comprise the minimum force template.

6 Conclusions

To move toward creating a JBI implementation, a thorough understanding of the domain of the JBI is necessary. The preceding discussion has attempted to identify the significant technical problems associated with a JBI implementation, the users of JBI and the

requirements for JBI. This discussion is does not contain enough detail to move directly to implementation but is meant to facilitate further discussion of the domain.

JB1 Dictionary

aggregation fuselet: a fuselet that creates new objects at its level. Example: A “group level” fuselet may look at a several tanks (lower level objects) and suggest the creation of a tank group. [12, 73]

attributes: attributes of an information object. The information object then provides values for these attributes. [12, 85]

client: “a computer that uses the JBI as part of a mission” [12, 76]

connector: software that connects to a legacy system and provides the translation between JBI Object and the native legacy system format. Also referred to as a “wrapper”. [12, 82]

common searchable attributes: attributes which are searchable quickly by the publish / subscribe system that include object, object identifier, the creator and the geospatial-temporal region associated with the object. [12, 90]

Fusion: the process of turning information from input sources into knowledge. [12, 82]

fuselet: Software intermediaries that evaluate the state of the SCR. Fuselets can publish data, effect a change in the state of the SCR or do numerical calculations in response to incoming events. Currently, fuselets are viewed as scripts that interact with the publish/subscribe middleware. [12, 73], [12, 77]

Gateway: A link between two C² systems. A gateway may connect a JBI to another JBI or connect a JBI to another C² system from a coalition partner [12, 83].

GOTS: government off the shelf

information object: data and recorded information. [12, 84] “Every JBI object is an instance of an *object schema*, which in other areas might be called a “class,” ...” [12, 85]

JB1 Administrators: “the information staff who work with the JBI to stand it up and operate it.” [12, 76]

JB1 Platform: The middleware foundation that supports fuselets and the publish-subscribe mechanism.

level 1 fusion: the perceptual indication an object is present. The physical knowledge of a battlespace object. [12, 72]

level 2 fusion: “context” and qualitative information about JBI objects. [12, 72]

level 3 fusion: inferring intent (specifically enemy intent: movement, plans, tactics, strategy, etc) from Level 2 information.

level 4 fusion: collection management. Level 3 intent data is used to suggest new sensor taskings for the purpose of improving the SCR. [12, 72]

metadata: attributes that describe the information object itself rather than real-world data. [12, 86]

object schema: definition of an object's attributes. An object schema defines a class of objects.

object identifier: a unique identifier for an object. Similar to a handle. [12, 87]

ontology: “the JBI’s vocabulary.” [12, 66], a dictionary used by the SCR so that objects can have semantic meaning, “If there is not an ontology entry, the object or concept cannot enter the SCR.”

pedigree: the other objects and processes from which the information object was derived. [12, 88]

proxy: a computer that will provide JBI services for devices that cannot interact with the JBI. This may be due to insufficient computational power on the device or because it is not accessible using TCP/IP protocols. [12, 83]

Structured Common Representation (SCR): The current understanding of the military situation within the JBI. [12, 66]

server: “a computer that provides platform services necessary for the JBI to run.” [12, 76]

users: “people who work with the JBI to exploit it” [12, 76]

References

- [1] Brayer, Kenneth “Global Grid for Large C2 /ISR Aircraft with Joint STARS Example”, Mitre Technical Report http://www.mitre.org/support/papers/tech_papers_01/
- [2] Carter, Harold W. “A Look into the Future of Warfare: The Joint Battlespace Infosphere”, Slides, University of Cincinnati No date available but slides obtained May 31, 2001 <http://www.ececs.uc.edu/~hcarter>
- [3] Dolphin Technology, Inc., “Joint Battlespace Infosphere (JBI) Concept of Operations (DRAFT)”, Slides, No date available but slides obtained on May 1, 2001
- [4] Information Directorate of the Air Force Research Laboratory (AFRL) “Joint Battlespace Infosphere (JBI) and Fuselet Research and Development”, Web Site: <http://www.rl.af.mil/div/prda/prda0009.html>
- [5] Odyssey Research Associates, “Consistent Battlespace Picture (CBP) Security Architecture, Final Report”, Purchase Order No. 00000037394 TR-01-0001, January 2, 2001
- [6] Pellisier, Stephen “A Brief Overview of Software Agent Applications and Risks”, SANS Institute, Information Security Reading Room, November 10, 2000, http://www.sans.org/infosecFAQ/casestudies/agent_apps.htm
- [7] Renner, Scott “The Joint Battlespace Infosphere FAQ”, Draft Version 4, December 5, 2000 Author email: sar@mitre.org
- [8] Renner, Scott “Explaining the JBI Reference Architecture: A Series of Lies”, Slides obtained from Mitre, unknown creation date but obtained on May 1, 2001. Author email: sar@mitre.org
- [9] Saunders, Thomas “A Vision of the Future for the Air Force (and perhaps the whole DoD)”, Mitre Technical Report, http://www.mitre.org/support/papers/tech_papers_01/
- [10] Surer, Julie “JBI” Slides, October 26, 2000, Mitre
- [11] Tracz, Coglianese, and Young. “Domain Specific SW Architecture Engineering”, *Software Engineering Notes*, 18(2), 1993.
- [12] United States Scientific Advisory Board, “Building the Joint Battlespace Infosphere” SAB-TR-99-02, December 17, 2000, <http://www.sab.hq.af.mil/Archives/index.htm>
- [13] United States Scientific Advisory Board, “Building the Joint Battlespace Infosphere: Volume 1: Summary” SAB-TR-99-02, December 17, 2000, <http://www.sab.hq.af.mil/Archives/index.htm>

[14] United States Scientific Advisory Board, “Building the Joint Battlespace Infosphere: Volume 2: Interactive Information Technologies” SAB-TR-99-02, December 17, 2000
<http://www.sab.hq.af.mil/Archives/index.htm>

APPENDIX B

“The Application of Probabilistic Risk Analysis to Security”

PROBABILISTIC RISK ASSESSMENT OF CRITICAL NETWORKED INFORMATION SYSTEMS

John C. Knight

Robert Schutt

(knight, rschutt)@cs.virginia.edu

March, 2002



Department of Computer Science
University of Virginia
151 Engineer's Way
PO Box 400740, Charlottesville, Virginia 22904-4740
(434) 982-2216

PREFACE

This report documents a research project that investigated the application of some of the techniques of probabilistic risk analysis to large, critical networked information systems. The goal was to see whether techniques developed for and successfully applied to system safety analysis could be applied to a different type of critical system—large networked information systems.

Such systems have become common and critical both for civilian and military users, and there is no existing comprehensive approach to the dependability analysis of such systems. Of special and primary interest was the application of probabilistic risk analysis to *security*. Present security failures in networked information systems are frequent and serious.

Two techniques from probabilistic risk analysis were explored. They were hazard analysis and fault-tree analysis. For purposes of experimentation, both were applied to one specific networked application, the U.S. Air Force's *Joint Battlespace Infosphere*.

This report is divided into two parts. Part one is a summary of the key ideas in probabilistic risk analysis together with details of extensions developed for this project. This part provides the background details necessary for complete understanding of the analysis. The analysis itself is presented in part two.

This work was funded in part by the United States Air Force under grant number F30602-01-1-0503.

PART I

THEORY AND DEFINITIONS

PROBABILISTIC RISK ASSESSMENT OF CRITICAL NETWORKED INFORMATION SYSTEMS — *THEORY AND DEFINITIONS* —

I-1. Introduction

Critical information systems provide service to applications, such as transportation, finance, and many military systems, where the consequences of failure are serious. Such information systems are usually networked, and they support a variety of important application services.

Systems such as these are expected to be *dependable*. We define this term precisely later in this document but, informally, the user expects that the system will be operational when needed (availability), that the system will keep operating correctly while being used (reliability), that there will be no unauthorized disclosure (confidentiality) or modification (integrity) of information that the system is using, and that operation of the system will not be dangerous (safety). Precise definitions of these terms have been developed over a period of many years by a group of researchers led by J.C. Laprie [2]. The notion of dependability is broken down into six fundamental properties: (1) reliability; (2) availability; (3) safety; (4) confidentiality; (5) integrity; and (6) maintainability.

It is important to note that security [1] is not included explicitly in this list. Rather, it is covered by the second, fourth and fifth items in the list—availability, confidentiality and integrity. The reason for this approach is that security is a compound property. In other words, requiring that a system be secure means that the system has to have more than one fundamental property. Confidentiality and integrity are concerned with the more traditional notion of security whereas availability is the property that is lost during a denial-of-service attack. Since such attacks are malicious, they are usually considered a security problem.

Some aspects of dependability are not completely intuitive. It is possible for a system to possess one property and not another although this is not reflected in the common use of the terms. For example:

- A communications system can be highly available because it provides service most of the time but unreliable because it actually fails for brief periods quite frequently.
- An aircraft system can have low availability because it is not ready for service most of the time but be safe because it does no harm.
- An information system can be reliable because it supplies continuous service yet insecure because it does not maintain the confidentiality of its content.

These notions are quite general and informal, and they must be refined into precise statements before progress can be made in creating systems that provide satisfactory service. Acceptable availability to one user might not be acceptable to another. Similarly, ensuring that information is not disclosed depends on the threats that occur. Casual hacking can be prevented quite easily but preventing disclosure by a determined insider with access to passwords is much more difficult.

In general, a perception that service is “obviously acceptable” is not sufficient. In order to be confident that a system will meet the needs of its users, it is necessary to define the system’s dependability requirements very carefully and then make sure that they have been met by the system being deployed.

In this report, we review the precise notion of dependability in critical information systems and investigate the analysis of the design of such systems. We explore the possible application of a technique called *probabilistic risk analysis* [12] that has been developed primarily to analyze safety systems to critical networked information systems, particularly the security of such systems.

There is a great deal of similarity between the subtle defects that lead to safety failures in critical systems and the subtle vulnerabilities that lead to successful security attacks. It is the claim of this research project that the application of probabilistic risk analysis (PRA) can help to deal with this problem by discovering systematically security and other vulnerabilities that would otherwise remain hidden or at least be very difficult to detect.

To investigate this claim, we have applied some of the techniques from probabilistic risk analysis to critical information systems. In order to have a specific target for experimentation, we have examined *publish/subscribe* systems using the Department of Defense’s *Joint Battlespace Infosphere* (JBI) as an important example.

Security of critical information systems is an urgent and difficult problem. Many successful attacks occur because of vulnerabilities of which the system’s designers are unaware. Vulnerabilities arise frequently because of *combinations* of circumstances, most of which are insignificant in isolation. Security analysis of critical information systems, if conducted at all, is often limited to audits of software and systems using checklists based on known problems. This approach provides very little assurance of satisfactory performance.

Probabilistic risk analysis is quite general, and we show that it can be applied successfully to critical information systems. As part of this application, we introduce some new concepts that facilitate the use of these techniques for security analysis. We conclude that this application yields results that are significantly more general in the security field than the techniques which are employed typically.

I-2. System Dependability

The first step in achieving dependability is to establish what the *dependability requirements* are for a specific system. Without properly stated requirements, any level of dependability has to be viewed as acceptable because there is no way of determining when service is unacceptable. Stating dependability requirements is quite a difficult task because it is influenced by many different aspects of the application.

A second difficulty is that there is no way to achieve perfect dependability (or even to come close to it), and so there is no point in requiring it. It is impossible to build systems that will operate without failure just as it is impossible to build systems that are totally secure. Any requirements statement must be achievable. We discuss how to do this for critical information systems using the JBI as an example.

The second step in achieving dependability is to engineer a system to meet the dependability requirements. This too is very difficult and for a wide variety of reasons. The major reason that concerns us in this paper is the difficulty of *analyzing* a design to show that it meets whatever dependability requirements have been stated. Assuring that a system design will provide some high level of availability, for example, is complex, and there are aspects of the problem for which no solution exists. There is no generally agreed approach to predicting the failure rate of software, for example, and so any quantitative analysis of a complete system has to be conditional on an assumed failure rate for any software that is used. Even if a simple life-testing approach is used for analysis of software, it is frequently impossible to carry out sufficient tests to be sure that a required threshold has been achieved [3].

Fortunately, many valuable forms of analysis can be carried out for system designs so that it is possible to show, for example, that events such as certain power failures, computer hardware failures and so on will not (or will) affect availability.

For security, the situation is even worse than with availability. Assuring that a system design will provide some requisite *quantified* level of security is essentially impossible because probabilistic quantification of threats and vulnerabilities is essentially impossible. What this means is that statements such as “this system is 99.999% secure” are, unfortunately, quite meaningless. We can, however, undertake *qualitative* analysis of systems from a security perspective and show, for example, that certain types of attack will not succeed, that certain types of vulnerability exist, and so on.

The set of techniques known as Probabilistic Risk Analysis (PRA) has been developed to try to permit analysis of the designs of complex systems [12]. PRA includes many techniques such as hazard analysis, fault-tree analysis, failure-modes-effects-and-criticality analysis, event-tree analysis, and hazard-and-operability analysis. Each of these techniques plays a role in helping to analyze the design of a complex system, in particular with regard to the system’s safety. In many cases, quantification of the probability of an accident can be estimated using these techniques provided assumptions are made about the

rate at which components fail, about the faithful implementation of a design, and about the completeness of the analysis.

This technology has been applied widely in many fields including the aerospace, defense, medical, and nuclear industries. The results have been impressive, allowing designers to have a high level of confidence that their systems are capable of meeting a variety of dependability requirements.

With this in mind, we have conducted a research project to apply some of the techniques of PRA to critical information systems. In principle, PRA should prove beneficial in many areas of the dependability analysis of critical information systems. In general, for example, it should be able to show where a system might be subject to problems such as the following:

- Power failure because of a power source that is a single point of failure.
- Data loss through inadequate provision for backup.
- Loss of service through physical damage by a terrorist.
- Loss of service through cable damage from a backhoe

Among the major benefits of the traditional application of PRA are the following:

- The process forces the analyst to *think* systematically about the design and possible faults in the system of interest. Although this is not a formal process, it has been demonstrated to be vastly superior to unorganized, ad hoc processes.
- The process forces the analyst to *document* key aspects of the system of interest systematically so that all the issues are in one place and properly organized.

Of particular interest in this research was the application of these PRA techniques to the security aspects of dependability. Although PRA has been applied widely to physical systems and embedded computer systems, its formal application to security issues is almost non-existent.

This project was undertaken to answer three research questions:

1. To what extent is PRA applicable to critical information systems?
2. For the special case of security, does PRA permit a more comprehensive determination of vulnerabilities than would otherwise be possible?
3. When applied to critical information systems, what system design changes are suggested by PRA (if any) that will allow such systems to meet more stringent dependability requirements?

Given these questions, our hypotheses on this topic are the following:

The techniques of probabilistic risk analysis, perhaps with enhancements, can be applied successfully to critical networked information systems and useful conclusions drawn.

Applying probabilistic risk analysis to the security of critical networked information systems will provide the benefits of systematic analysis and documentation as occurs in safety analysis. The resulting analysis and documentation will be superior to those typically available via ad hoc security analysis techniques.

The project described here is limited to the application of two components of PRA: (1) hazard analysis; and (2) fault-tree analysis. Other techniques will be explored in future research. These two techniques were selected because they are among the most commonly applied components of PRA, and because their application seemed most likely to be successful.

I-3. Dependability Concepts

In this section, we summarize the essential definitions from the field of dependability. Many of these definitions are from the Laprie framework, and they are precise and testable [2]. With these definitions in hand, it is possible to determine how they can be applied to critical information systems, and to create a precise framework for documenting dependability requirements.

I-3.1 Faults, Failures, Errors and Hazards

Before discussing the fundamental dependability properties, it is necessary to review what causes a system *not* to be dependable. If a system were perfect and stayed that way, it would never fail to supply acceptable service, and there would be no concern for its performance. However, systems do fail, they do break as time passes, and in some cases, they contain defects in their basic design. These thoughts require a precise definition of the notions of *failure*, *error*, and *fault*.

A *failure* has occurred when a system no longer complies with its requirements. Since requirements only specify the external behavior of a system, the notion of failure is only relevant to what the system's users experience. This is an important idea because the failure of components within a system is both expected and possibly common. The trick to dependable design is to make sure that failed components within a system do not lead to system failure.

It is useful when talking about dependability to separate failures into those that are detectable and those that are not. Undetectable failures tend to be more serious because users cannot take account of the problem. For example, if security penetration is detected, all sorts of corrective and protective actions can be taken. If it is not detected, users continue

to trust the system even though this trust is misplaced. Similarly, if a major control surface fails on an aircraft and the pilot knows about it, he or she can take corrective action. If the failure of the control surface is not known by the pilot, any actions that he or she takes will be based on significant misinformation and the likelihood of a crash is much higher.

An *error* is a system state which, if nothing is done to prevent it, might lead to a system failure. Thus, for example, a system state in which various data elements are corrupted for some reason might well lead to a system failure unless the system detects and corrects the error. It is important to remember, therefore that *failure* is when an *error* reaches the service interface. Errors that do not reach the service interface are not relevant to the user because they do not impact the provided service. Thus they do not affect the dependability of the system.

A *fault* is the adjudged cause of an error. Informally, a fault is something that has gone wrong. A defect in a piece of software that causes some of the system's data elements to become corrupted is a fault. Similarly, a computer that malfunctions is a fault, as is a break in a power or communications cable caused by an errant backhoe.

There are two basic types of fault—*degradation* faults and *design* faults. Degradation faults have the property that they arise over time, and a component that operates correctly for some period of time might stop doing so as a result of a degradation fault. The simplest example of a degradation fault is when the filament in a light bulb breaks. The light bulb was working but ceases to do so when the filament breaks. In essence, the filament has worn out.

A design fault is something that is wrong with a system which has been there from the outset. It is not the result of a component wearing out. All software faults are design faults because software does not degenerate in the usual sense. However, design faults can be present in any element of a system's design.

Hazards are system states that could lead to undesirable outcomes which result from using a system. The primary danger in most medical systems, for example, is patient injury, and so any state in which a patient might be injured is a hazard. Note that a hazard is just a system state. In particular, a hazard is not an accident. A hazard is a state from which an accident (or some form of loss) might occur.

Hazards are not obvious, and determining the hazards for a given system is often a difficult and complex task. In a system designed to irradiate a patient, for example, one danger might be to deliver too much radiation. A second, much less obvious hazard would be to deliver too little radiation because in that case the treatment might be less effective than was expected.

A hazard can arise when a fault is manifested and the effects of that fault reach the environment in which the system is operating. In many cases, the manifestation of a fault does not lead to a hazard because the effects are masked in some way. Such faults are said to have been *tolerated*. This is the basic principle underlying so-called fault-tolerant systems.

I-3.2 Reliability

Reliability of a system is the property of continued correct service. It is expressed as a probability that the system will continue to operate correctly. More precisely, for a given system and a given operating environment:

$$\text{Reliability}(t) = \text{probability}(\text{system operates correctly up until time } t)$$

A reliability (probability) of 0.999 for a time t of ten hours means that, for the specified operating environment, the probability that the system will operate correctly for ten hours is 0.999. Referring to a system as *reliable* means that it achieves this probability in service. It does not mean that the system never fails nor that it fails infrequently. This is akin to the observation that it is perfectly possible to get a string of ten heads when tossing a fair coin.

I-3.3 Availability

Availability of a system is the property of readiness for service. It is expressed as a probability that a system is available for use over some time period. More precisely, for a given system and a given operating environment:

$$\text{Availability} = \text{probability}(\text{system ready for service at time } t, \text{ for all } t)$$

A system availability of 0.999 means that, at any given time, the probability that the system is ready for use is 0.999. Again, referring to a system as *available* means that it achieves this probability. It does not mean that the system does not fail nor does it mean that the system fails infrequently. A system that failed for a period of one millisecond every ten seconds, continuously, would have an availability of 0.999.

I-3.4 Safety

Safety of a system is the property of doing no harm, i.e., that the system operates without catastrophic consequences. Safety means that the probability that a system will cause harm greater than a certain value is less than a level acceptable to users. The notion of value is purposely vague to include number of human lives, dollars lost or any notion of what is valuable to the community.

If certain catastrophic consequences are known to be possible and for each such consequence a value can be associated with the resulting damage, then if probabilities for such events can be estimated, an expected loss from the operation of the system can be calculated. This expected value of loss is known as the *risk* associated with using the system. A system is safe in a formal sense if the risk of operating the system is below some prescribed threshold.

I-3.5 Integrity

Integrity is the property of absence of improper state alterations. Of most interest and concern are malicious state alterations since this is a major element of security. If an adversary

has the ability to change a systems state, the subsequent actions of the system cannot be trusted.

This property is concerned with the entire state of a system including, for example, all user data and operating system tables. Unlike the previous three properties, there is no general notion of probability associated with integrity. The system state is either as it should be as a result of system operation or unauthorized state changes have been made.

I-3.6 Confidentiality

Confidentiality is the property of no unauthorized disclosure of information. This property is concerned also with the entire state of a system. Again, there is no general notion of probability associated with confidentiality.

I-3.7 Maintainability

Maintainability is the property of a system of having the ability to undergo repairs and modifications. This property is important for achieving the other properties defined in this section because often the operating environment for a system changes, and, in order to continue to meet the dependability requirements, the system must be updated. To identify the maintainability requirements, it is important to identify possibly updateable components.

I-4. Probabilistic Risk Analysis

In this section we summarize the two techniques of interest—hazard analysis and fault-tree analysis. The combination of the two provides an approach to the modeling of faults in critical systems, and the combination is often used to assess the designs of safety-critical systems. The assessment is quantitative if probabilities are available for the complete sets of events documented in the fault tree. Further details of these techniques can be found in the texts by Modarres [12] and by Storey [21].

I-4.1 Hazard Analysis

Hazard analysis is the process of determining the hazards to which a specific system will be subject. Inevitably, hazard analysis is an informal process because that which constitutes a hazard is based on human perceptions and values. Although it is informal, it provides a framework for rigorous analysis. By forcing the developers of a system to think systematically about the states in which severe damage might occur, it provides a means of exploring the space of potential damage and yields far more comprehensive results than ad hoc methods.

There are some standard techniques and procedures for hazard analysis that are based on systematic reviews of some aspects of a problem domain. For example, in systems that employ some form of energy, say electrical energy, a hazard analysis can in part be guided

by the physical sources of energy and the need to contain it. There is a striking similarity here between sources of energy and sources of information.

As well as being informal, hazard analysis is very dependent on the details of an application and the associated domain. Even for apparently similar applications, what constitutes a hazard for one might not be for another. Thus, an important first step in performing a hazard analysis is to bound the system being considered.

The challenge in hazard analysis is to determine which states might arise that could result in serious negative outcomes. These states are determined by analyzing the *consequences of failure* for the system. If there are consequences of failure that are considered by the systems' users to be unacceptable, then a hazard is any state that could lead to that failure. The process of hazard analysis for a general system can be carried out by determining the consequences of failure that are unacceptable and then determining the states from which such failures might occur.

As an example, consider a device designed to provide some form of medical treatment that is not required to be administered in an emergency situation. A state in which the device is not functional is almost certainly not a hazard because the consequences of failure are merely inconvenient. For the same device, a state in which the wrong treatment could be administered is clearly a hazard because the consequences of failure include patient injury.

I-4.2 Fault-Tree Analysis

Determining the faults that might occur and what the effects of a fault will be is the role of *fault-tree analysis*. It is usually the case that a separate fault tree is developed for each hazard that has been identified. The purpose is to explore the system design and determine the complete set of faults that could lead to that particular hazard. The term fault tree is a little misleading since the tree is composed of nodes that represent *events*. An event in this case corresponds to the manifestation of a fault.

As with hazard analysis, the creation of a fault tree is informal. Once again, however, it provides a framework for rigorous analysis and yields far more comprehensive results than ad hoc methods. The development of the fault tree for a specific hazard forces engineers to try to enumerate all the circumstances that will lead to the hazard, and this activity is the most rigorous process available for determining the dependability of a system design.

The various forms of analysis to which fault trees can be subjected are completely formal. The best known form of analysis is the calculation of the probability of the hazard to which the fault tree applies from the probabilities of the various leaf events.

Additional rigor can be introduced into the creation of fault trees by including techniques such as Failure Modes, Effects and Criticality Analysis (FMECA) and Hazard and Operability Analysis (HazOp). These techniques force the analyst to think more carefully about

the possible effects of component failures (FMECA) and about possible system perturbations (HazOp).

Fault-tree analysis begins by identifying all of the events that could lead directly to the hazard at its root. If there is more than one such event, then the tree is built to represent the logical combination of events that could lead to the hazard. If either of two events could lead to the hazard, then both are shown and they are connected to the hazard (root) using an OR operator. If both of the events are required, then both are shown and they are connected to the hazard using an AND operator. The symbols used for logical operators are usually the gate symbols from digital circuit design.

Development of a fault tree proceeds by recursively determining the events that could lead to the existing leaf nodes in the tree, and continues until events are identified that can be considered basic.

I-5. Previous Work

The use of formal analysis of the type proposed here has received only scant attention in the literature. Previous work in the areas of vulnerability analysis, argument trees, insecurity analysis, attack patterns, attack graphs, and attack trees is discussed in this section.

All probabilistic risk analysis techniques (PRA) focus on identifying sequences that could lead to a system failure. Two main types of analysis have been examined for potential use in vulnerability analysis: human based techniques that are systematic approaches to discovering vulnerabilities and computer based techniques that allow for the automated enumeration of the search space. Both human-based and automated systems have their strengths and limitations. The results from human-based probabilistic risk assessment are often criticized as being incomplete and error prone [7]. Fully automated systems based on attack graphs are problematic as it has been shown that attack graph generation is NP-complete [15].

I-5.1 Methodically Organized Argument Trees

Keinzle and Wulf proposed Methodically Organized Argument Tree (MOAT) as a technique for aiding in developing informal proofs of system security properties [8]. A MOAT is for all systematic purposes a fault tree except that the root of the tree is not a negative outcome but rather a positive desirable property for the system. From the root down, the analyst proceeds by identifying everything that must be maintained for the higher level property to continue to be true. Any MOAT can be converted to an equivalent and equally expressive traditional fault tree through simple application of DeMorgan's Law. A traditional fault tree has the negative outcome at the root whereas a MOAT has a positive desirable property as the root. To obtain the fault tree corresponding to the MOAT using Boolean logic $!(MOAT) = Fault\ Tree$. The result of this negation is that each node is negated and AND gates are exchanged for OR gates and vice versa. According to Kienzle and Wulf, the value of a MOAT is that it may not be easy to formally prove the entire

security property, but it might be possible to develop formal proofs for individual nodes in the MOAT. This allows for various elements in the system to be analyzed with different levels of detail. Kienzle and Wulf suggest that analysts will want to explore “high risk nodes” more carefully than others. The MOAT facilitates this exploration by allowing the analyst to leave low risk nodes with rough security proofs while fully specifying proofs for other higher risk nodes.

I-5.2 Attack Trees

Almost all successful human-based probabilistic risk assessment techniques have used fault trees to some extent in system analysis. Fault trees have existed since the early 1960’s and have been used extensively in the design of safety-critical systems. NIST has published recommendations on the use of fault trees in security analysis of PBX systems [22]. In addition, the FAA has outlined the use of fault tree analysis as a technique for hazard analysis in aircraft systems [5].

As previously described, fault trees rely on the assembly of Boolean gates to form a logical hierarchy. Recently, Schneier has shown how fault trees can be used as a technique for system security analysis [20, 19]. Although Schneier calls his analysis technique “attack trees,” they are identical in every regard to fault trees. Attack trees (fault trees) prove to be very interesting and useful in analyzing system security. Schneier demonstrated that each leaf in a fault tree can have an associated value and these values can be propagated up the tree to determine the cost of an attack. This cost parameter may be measured using any metric but dollar value or time are particularly useful for security analysis. After assigning the associated costs to the leaves in the tree and propagating their values up the tree, it is possible to algorithmically sort possible attacks according to cost. This output is useful for identifying low cost attack paths and prioritizing implementation of security features.

I-5.3 Attack Patterns

One problem with fault trees is that, since they require extensive human interaction to construct, development of a fault tree is a labor intensive process. Linger and Moore expanded on Schneier’s re-application of fault trees and provide a framework for the reuse of attack trees in system vulnerability analysis [13, 11].

Linger and Moore introduced the concept of attack patterns and attack profiles as complementary elements to fault tree analysis [11]. As Moore, Ellison and Linger have described it, an attack pattern is a sequence of steps in an attack [13]. The sequence of steps in the attack pattern is a fault tree. In addition to the fault tree an attack pattern includes: a defined purpose, a set of pre-conditions, and a set of post-conditions. Since the sequence of steps in the attack is now parameterized by the pre-conditions and post conditions, it is possible to reuse the attack pattern in a different context provided the pre-conditions and post-conditions are satisfied. Moore, Ellison and Linger call a set of an attack patterns and variants for those patterns an attack profile. Attack profiles are generic sets of attack patterns and can be instantiated with specific values for the system being analyzed. Moore, Ellison and Linger claim that through the use of attack patterns and attack profiles, a

library of attack trees can exist and provide reuse much in the way functional procedures provide for code reuse in software engineering.

I-5.4 Attack Graphs

Attack graphs are an alternative method of system analysis that uses an automated process to identify system vulnerabilities. One of the early successful systems was designed by Phillips and Swiler at Sandia National Labs [15]. Their system uses a state machine to represent the objects (computers, network links, components, etc.) and transitions are possible actions with those object. The objects and transitions form a state machine. Using this state machine, it is possible to automatically search for a sequence of possible events (set of state transitions) that will result in a negative outcome (hazard). As an example if the state Z is the hazard, and S is the start state, and valid transitions are: $S \rightarrow X$, $X \rightarrow Y$, $Y \rightarrow Z$ it is possible to search this space algorithmically to determine that starting in state S , applying $S \rightarrow X$ then $X \rightarrow Y$ followed $Y \rightarrow Z$ will enter the hazardous state.

It is interesting to note that there is a close correlation between the output of an attack graph generator and a human generated fault tree. Each sequence of transitions that leads to the overall negative outcome is one path to the root in a traditional fault tree. The obvious advantage to attack trees is that the automated process can completely search the state space. In addition, various graph algorithms such as the shortest path algorithm can be applied to extract the least cost set of transitions after the attack graph is generated. The effectiveness of attack graph generation is dependent on how closely the state machine and transitions represent the real system and possible actions. If an object or possible transition is not identified in the input, it is possible that a system vulnerability will not be identified by the process. Other researchers have worked on technologies closely related to attack graphs including Moskowitz and Kang [14], Ritchey and Ammann [17], Sheyner et al [18], and Jha et al [7]. Most of this work has focused on demonstrating how model checking techniques can be used to identify paths in an attack graph. Existing model checking systems provide many of the mechanics necessary for attack graph analysis without significant modification since model checking and attack graph generation are algorithmically equivalent problems.

I-6. Systematic Security Analysis

The previous work in this area has provided some useful insights but has not applied the concepts of probabilistic risk analysis comprehensively. In this section, we introduce *systematic security analysis*, a technique that includes: (1) an extended form of classic hazard analysis called *information hazard analysis* and (2) an extended form of the classic fault tree called the *security-enhanced fault tree*.

Taken together, these two techniques provide a basis for rigorous security analysis of information systems. The results are similar to those obtained for safety-critical systems using traditional probabilistic risk analysis.

I-6.1 Information Hazard Analysis

Information hazard analysis targets the flow of information within a system, and identifies hazards based on the prospect of losing either confidentiality, integrity or availability of information. Thus the process is similar to that undertaken with hazard analysis in safety systems in which the undesired release of energy is the concern.

The information hazard analysis process is easy to state but difficult to achieve in a thorough and comprehensive way. The process is:

1. Determine all types of information within the system.
2. For each type of information, determine its criticality to the system's stakeholders.
3. For each type of information, determine the set of operations that can be carried out on that type of information during normal system operation.
4. For each type of information and for each possible operation on that type of data, determine whether the combination is of sufficient importance that loss would be considered catastrophic.
5. For the set of data type and operation combinations whose loss would be considered catastrophic, determine the set of system states that could cause the loss. These are the hazardous states of interest.

I-6.2 Security-Enhanced Fault Trees

The goal with a security-enhanced fault tree is to introduce the major types of event that of concern in security analysis into the basic concept of a fault tree so as to permit both construction of the tree and analysis that is tailored to security concerns.

We have introduced four types of fault-tree node for purposes of documenting and analyzing security issues. These four node types are *threat event*, *vulnerability event*, *compromise event*, and *attack event* with the following meanings:

- *Threat Event.*
A threat event is any event undertaken by an adversary. A state-space search attack on an encrypted link is an example of such an event.
- *Vulnerability Event.*
A vulnerability event is an event that exposes a vulnerability in a system. Failing to encrypt a message prior to transmission over an untrusted link is an example of such an event.

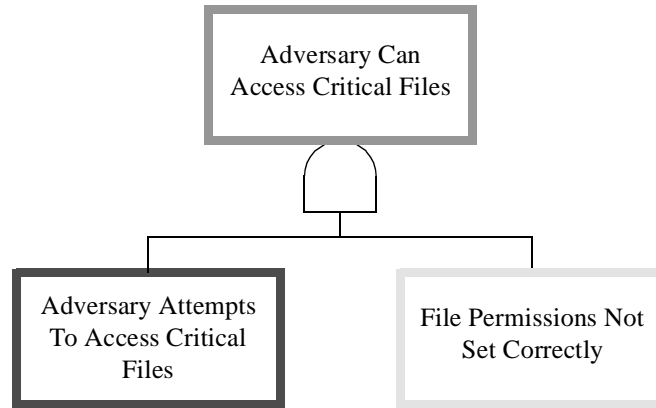


Figure 1. Security-enhanced fault tree.

- *Compromise Event.*
A compromise event is a compound event that results from the combination of a threat event and a vulnerability event. Although the hazard of interest might not have occurred, a state has arisen that could lead to that or another hazard.
- *Attack Event.*
An attack event is a compound event that results from one or more combinations of threat and vulnerability events and which results in the hazardous state.

In the graphic representation of the security-enhanced fault trees, these different node types are color coded as follows: threat nodes are shown in red; vulnerability nodes in yellow; compromise nodes in green; and attack nodes in white. This color coding allows the analyst to determine the general state of a system quite quickly.

Figure 1 shows a simple fault tree fragment with the three of the four types of event nodes. The threat is that an adversary attempts to examine critical files, the vulnerability arises because the files were stored with incorrect access permissions, and the compromise event is that the system has entered a state in which the adversary can read the critical files. In this simple case, this final event can be thought of as the hazard.

PART II

ANALYSIS OF A JOINT BATTLESPACE INFOSPHERE

PROBABILISTIC RISK ASSESSMENT OF CRITICAL NETWORKED INFORMATION SYSTEMS — ANALYSIS OF A JOINT BATTLESPACE INFOSPHERE —

II-1. Project Scope

Both the determination of dependability requirements and the analysis of a system design to determine whether dependability requirements will be met are goals that can only be undertaken for a *specific* system. In other words, when presented with a specific system including a complete specification of the operating environment, domain experts and systems engineers can determine the necessary dependability requirements for the system and analyze its design.

For our purposes, certain general statements could be made about applying PRA to critical information systems without specific system information, but, in order to be able to examine PRA comprehensively, we require a target for this examination. Rather than work with a particular system that happens to be available but would be atypical, we chose instead to use for analysis a generic system that exhibits the essential characteristics of a critical information system. We refer to this system throughout the remainder of this document as the *target system*, and we refer to its operating environment as the *target operating environment*.

The definition of the target system is problematic because it is not clear what level of detail should be included. We have dealt with this problem by defining the target system via a set of progressively more detailed refinements. Refining here means making assumptions about the system architecture and application, and so, as the refinements are developed the system becomes more and more specific. The goal, of course, is to minimize the level of detail and the number of assumptions made. The final refinement is shown in Figure 2.

The major architectural assumptions that are made about the target system are:

- *Publish/Subscribe Middleware*

It is assumed that the communications infrastructure that it uses is a publish/subscribe system. Publish/subscribe middleware is a promising technology for use in building information systems. The utility of this technology derives from its scalability, flexibility, and overall applicability [4]. Many information systems are being built or planned that use a publish/subscribe structure, and dependence on such applications will, in some cases, be considerable. Systems that provide user services in financial applications are examples from the civilian sector of systems whose proper functioning is critical and will become increasingly so.

- *Arbitrary Set of Publishers*
We assume that some large number of publishers is using the system and that the set of active users changes over time.
- *Arbitrary Set of Subscribers*
We assume that some large number of subscribers is using the system and that the set of active users changes over time.

The major application assumption that is made is that the application is a *Joint Battlespace Infosphere* (JBI). The Joint Battlespace Infosphere concept is an example of a class of military systems upon which a great deal of reliance will be placed. A JBI will be used to provide a variety of information services to a variety of military personnel. Since much of the information in a JBI will probably be classified, security is of paramount importance. Routine security audits are not sufficient in such cases because the prospect of undiscovered vulnerabilities is very high. We hypothesize that systematic security analysis as defined in part I of this report will enable many security issues to be determined that would otherwise be missed.

A complete analysis of a JBI including the determination of all the relevant hazards and the creation of the complete fault trees for all of the hazards is a major undertaking, and such an analysis is beyond the scope of this project. The primary research results of interest can be obtained with just part of this analysis, and, in this section, we document the analysis actually undertaken. This analysis consisted of:

- The determination of a representative set of hazards.
- Creation of a complete fault tree for one security-related hazard.

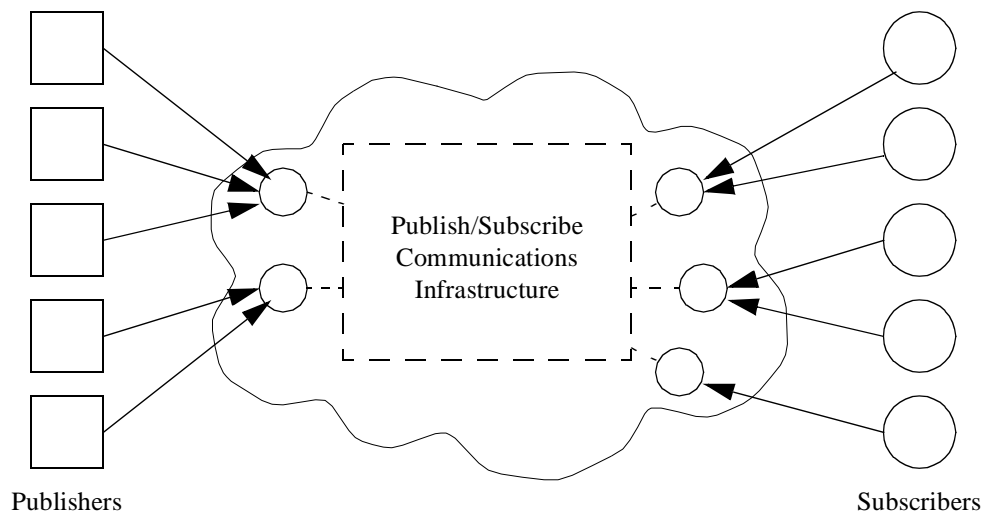


Figure 2. Joint Battlespace Infosphere Sample Architecture

- Creation of a set of common subtrees that are referred to by the sample complete fault tree.

II-2. Publish/Subscribe Primitives

In order to enable detailed analysis of the target system, it is necessary to analyze the assumed publish/subscribe communications architecture in detail. A set of publish/subscribe primitives is presented here that defines what will be considered the correct operation of a publish/subscribe system for the scope of this document.

II-2.1 Definitions

Publish/Subscribe System – A system that allows for the routing of information between clients. The routing of the information is determined by the content of the messages.

Routing Network – A non-empty set of routers connected in a network topology. The connection topology is at a minimum a spanning tree.

Routing Node – A single node in the routing network.

Client – The software that interacts with the publish/subscribe middleware. The API used by a client is the *service interface* for the system. Note, the client may only connect to one router node at a time.

Events – The encoding of information transmitted between clients in the system. Events are an aggregation of attribute/value pairs.

Attribute – A property that may take on a variety of values.

Value – The quantity that is bound specifically to an attribute

Pattern – A criterion that separates all events into one of two categories, events that match the pattern and events that do not match the pattern.

II-2.2 Publish/Subscribe Service Interface

The service interface is the application programming interface that client software uses to interact with the publish/subscribe system. In this section we define a hypothetical interface that does not represent any specific system. Many unimportant issues are omitted although they would be required in an operational system. Defining this interface carefully is important because it is this interface that is used in the documentation of hazards.

Connect (RouterNode n , Authorization A) – Client connects to the router node specified and presents their authorization to the system.

Subscribe (Pattern p) – After a client executes this command all future events published by any client that match Pattern p will be sent to the client.

Unsubscribe (Pattern p) – Client no longer receives future events matching Pattern p .

Publish (Event e) – Event e is injected into the routing network and transmitted to all clients that have subscribed with a pattern that matches Event e .

II-3. JBI Dependability Requirements

The dependability requirements of a given JBI will depend upon the multitude of characteristics for that specific JBI and its operating environment. It is probably the case that any given JBI will have dependability requirements somewhat like the following:

- *Availability.*
A very high level of availability will be needed since all users of a JBI expect to be able to obtain service upon demand. It is reasonable to assume that the specific probability would be 0.9999 or higher corresponding roughly to one hour of unavailability per year.
- *Confidentiality*
Any JBI will require the highest possible level of information confidentiality but no meaningful quantification of confidentiality is possible. We specify the confidentiality aspect of dependability as comprehensive analysis and documentation of the aspects of a system that might lead to a loss of information confidentiality.
- *Integrity*
As with confidentiality, any JBI will require the highest possible level of information integrity. Again, however, no meaningful quantification of integrity is possible. We specify the integrity aspect of dependability as comprehensive analysis and documentation of the aspects of a system that might lead to a loss of information integrity.

II-4. JBI Information Hazard Analysis

Given the theory and definitions of dependability summarized in part I and the concepts of publish/subscribe systems outlined in section II-2, we are now in a position to undertake the analysis that is desired. We begin in this section by determining the hazards to which the system will be subject. This analysis is based on the target system shown in Figure 2. In each case, the hazard is a state in which the event might occur.

II-4.1 Joint Battlespace Infosphere System—Consequences of Failure

The consequences of failure for a JBI are determined by the use to which the system is being put. We make the assumption that serious consequences could result from only the following three general states:

1. Functionality is unavailable. This state admits the possibility that critical information will not be available to those who need it.
2. Information is disclosed to persons without the requisite authority. This state admits the possibility that critical information will become known to an adversary.
3. Information is modified by persons without the requisite authority. This state admits the possibility that an adversary could invalidate previously sound information.

Unfortunately, these three state definitions are insufficiently precise. In the case of 1, for example, there are many forms of functionality for which loss would have little effect. For example, an inability to publish some forms of data by a single system user is probably not critical. Conversely, an inability to publish some forms of data by ALL system users is probably critical. Similarly, an inability to publish some forms of data by a key system user (such as a senior commander) might be critical.

Once again, we deal with this uncertainty by making assumptions about what would constitute a definitive set of failure consequences for a particular JBI. For any given JBI that is to be deployed, a careful analysis would reveal the detailed consequences of failure needed to guide the hazard analysis. The specific assumptions are:

- All published data is secret.
- Certain (unspecified) publishers are publishing information that is extremely critical. Thus a loss of JBI service to just a single publisher is a hazard.
- Certain (unspecified) subscribers are subscribing to information that is extremely critical. Thus a loss of JBI service to just a single subscriber is a hazard.

II-4.2 Joint Battlespace Infosphere—System Hazards

The specific hazards that we define for this example are:

Hazard 1: Publisher confidentiality violated.

The value of a published item is *revealed* to one or more unauthorized parties.

Hazard 2: Subscription confidentiality violated.

The value of a subscription is *revealed* to one or more unauthorized parties.

Hazard 3: Publication integrity violated.

The value of a published item is *changed* by one or more unauthorized parties.

Hazard 4: Subscription integrity violated.

The value of a subscription is *changed* to one or more unauthorized parties.

Hazard 5: Publication service unavailable.

One or more clients (but not all) unable to *publish* when necessary. Note that this includes the issue of *denial-of-service* attacks as a special case.

Hazard 6: Subscription service unavailable.

One or more clients (but not all) unable to *subscribe* when necessary. Note that, again, this includes the issue of *denial-of-service* attacks as a special case.

Hazard 7: JBI publication service unavailable.

All clients unable to publish.

Hazard 8: JBI subscription service unavailable.

All clients unable to subscribe.

II-5. Security-Enhanced Fault-Tree Analysis

For purposes of this research, we have developed a security-enhanced fault tree (see “Security-Enhanced Fault Trees” on page 67) for the first of the hazards identified in section. The fault-tree analysis is based on the target architecture and all the various essential assumptions described earlier in the document.

For the hazards associated with a major system of the type being discussed here, the fault trees can be *very* large. The size is determined by which faults are to be considered elementary in the analysis—the more elementary the item with which the fault is associated, the more elementary the fault. There can be hundreds, sometimes thousands, of elementary faults (leaves of the fault tree) and dozens, sometimes hundreds, of compound faults (intermediate nodes of the fault tree). It is common for a specific fault to appear more than once in a fault tree, and, for compound faults, this means that the subtree associated with that compound fault will be repeated.

Repeated subtrees occur in the publish/subscribe paradigm because the paradigm makes two significant assumptions:

1. There exists a computing platform on which clients and routers will run.
2. There exists a network for transporting messages between entities in the publish/subscribe system.

Defining faults related to these two assumptions is beyond the scope of fault-tree analysis of publish/subscribe middleware because they relate to problems that occur outside the service interface. However, most dependability problems occur because of errors at this

level that propagate through and manifest themselves in the publish/subscribe middleware. To accommodate this, common subtrees have been created to encompass these faults in the high level publish/subscribe fault tree. They are documented in section II-5.1.

II-5.1 Common Subtrees

Fault Tree: Network Compromise

The attacker is able to monitor and control the flow of traffic in the network connecting clients to routers and routers to routers. The type of compromise can lead most directly to a loss of confidentiality because the attacker can monitor network traffic between connected components. It can however lead to other problems if the software is updated over a network the attacker has compromised. In this instance, the software can be modified prior to its deployment during a system update.

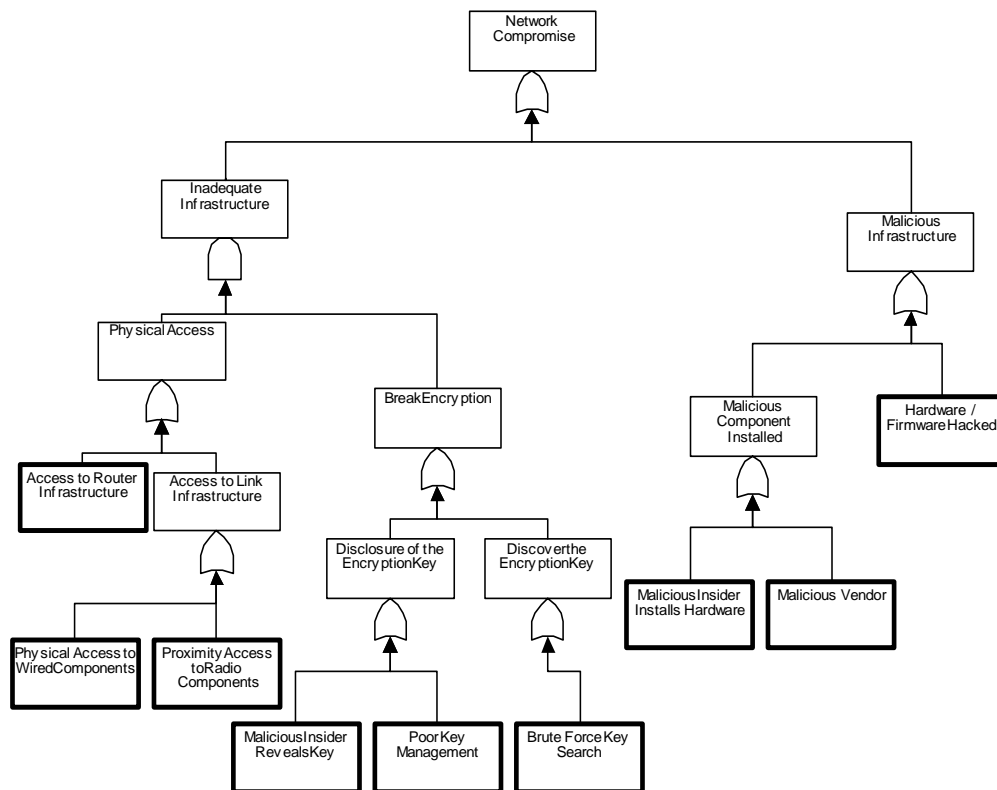


Figure 3. Network Compromise Fault Tree

Fault Tree: Computing Platform Compromise

The attacker is able to control all aspects of the computer on which a component of the publish/subscribe middleware is running on. This type of compromise can lead directly to a variety of failures including loss of availability, reliability, confidentiality, integrity, or maintainability because the attacker has complete control over the machine.

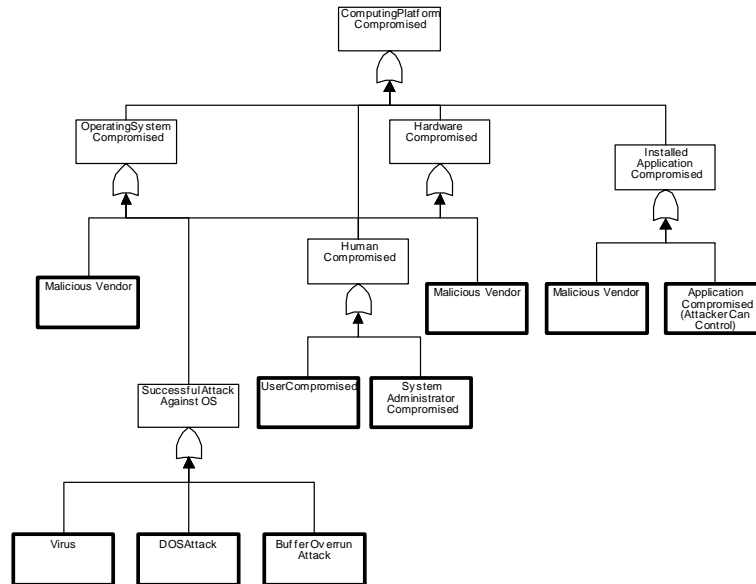


Figure 4. Computing Platform Compromised Fault Tree

Fault Tree: Development Process Compromised

The attacker affects the running components of the publish/subscribe middleware by compromising the software development process. It is assumed that a development process compromise can lead to a loss of availability, reliability, confidentiality, integrity, or maintainability because the attacker can direct the software to behave in any manner that is contrary to dependability goals.

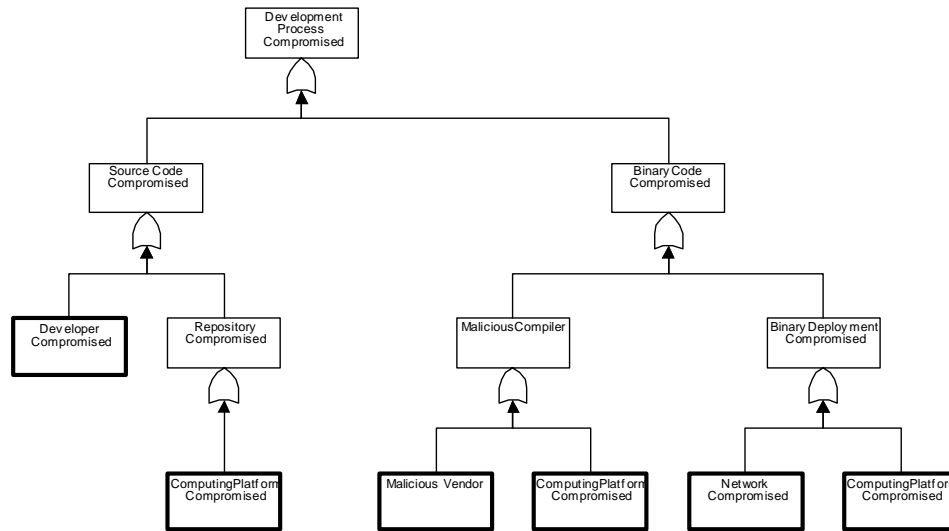


Figure 5. Development Process Compromised Fault Tree

Fault Tree: **Router Compromised**

The attacker compromises a router in the system and can control the actions of the router. A router compromise can effect publish/subscribe notifications because the router can effect and monitor traffic passed through it. In addition, loss of confidentiality and availability can occur because compromised router can disclose information and control the correct functioning of the service.

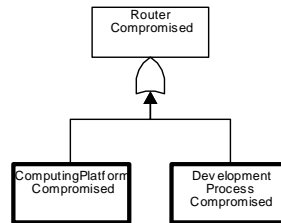


Figure 6. Router Compromised Fault Tree

Fault Tree: **Client Compromised**

In this instance, the attacker has compromised the publish/subscribe middleware aspect of the client software. A compromised client can lead to a loss of confidentiality because it can monitor in-bound and out-bound traffic between the client and the connected router.

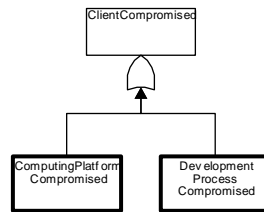


Figure 7. Client Compromised Fault Tree

Fault Tree: Development Process Problem

Problems with the development process can cause will eventually be reflected in the performance of the publish/subscribe system. The typical result of development process problems is software that is defective and either becomes unavailable during execution (crashes) or produces the wrong functionality (erroneous computation).

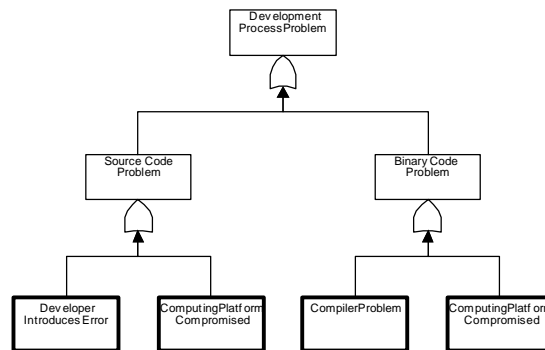


Figure 8. Development Process Fault Tree

Fault Tree: Computing Platform Unavailable

Client software is unavailable and no longer able to provide service.

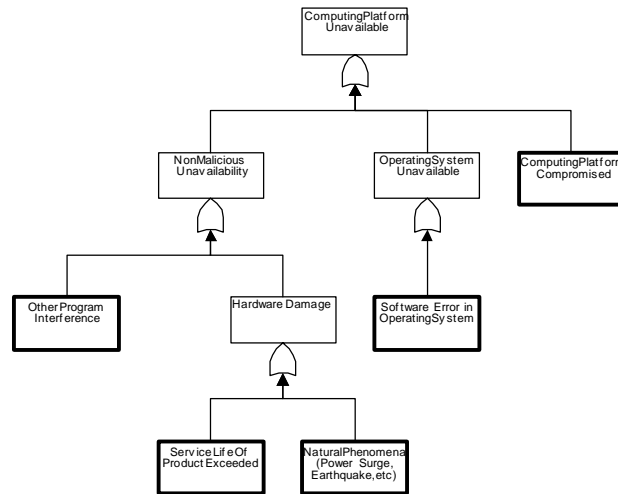


Figure 9. Computing Platform Unavailable Fault Tree

Fault Tree: Network Unavailable

The network used by the routers and clients becomes unavailable. Any unavailability of the network the publish/subscribe middleware uses as the transport mechanism for notifications is critical to the availability of the middleware itself.

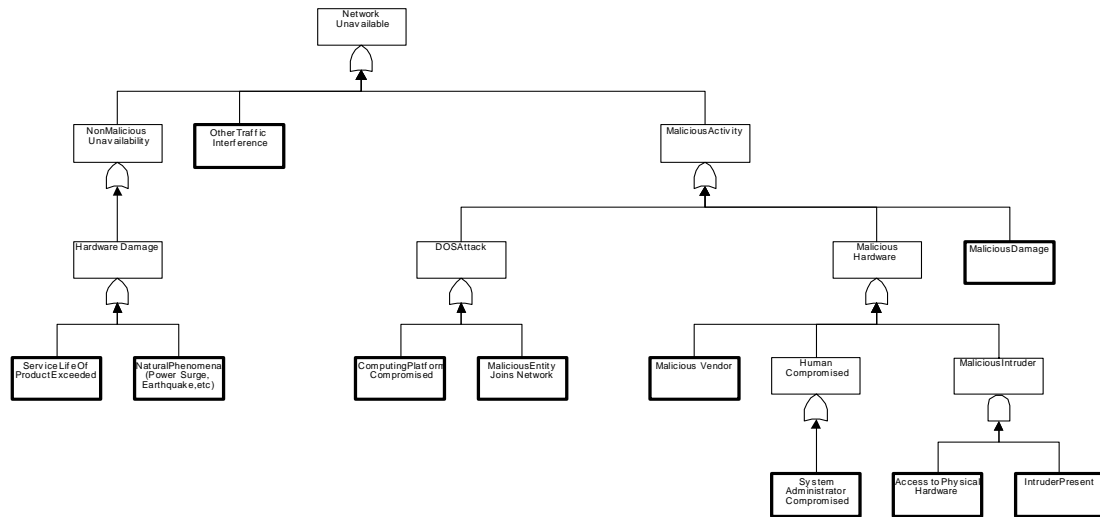
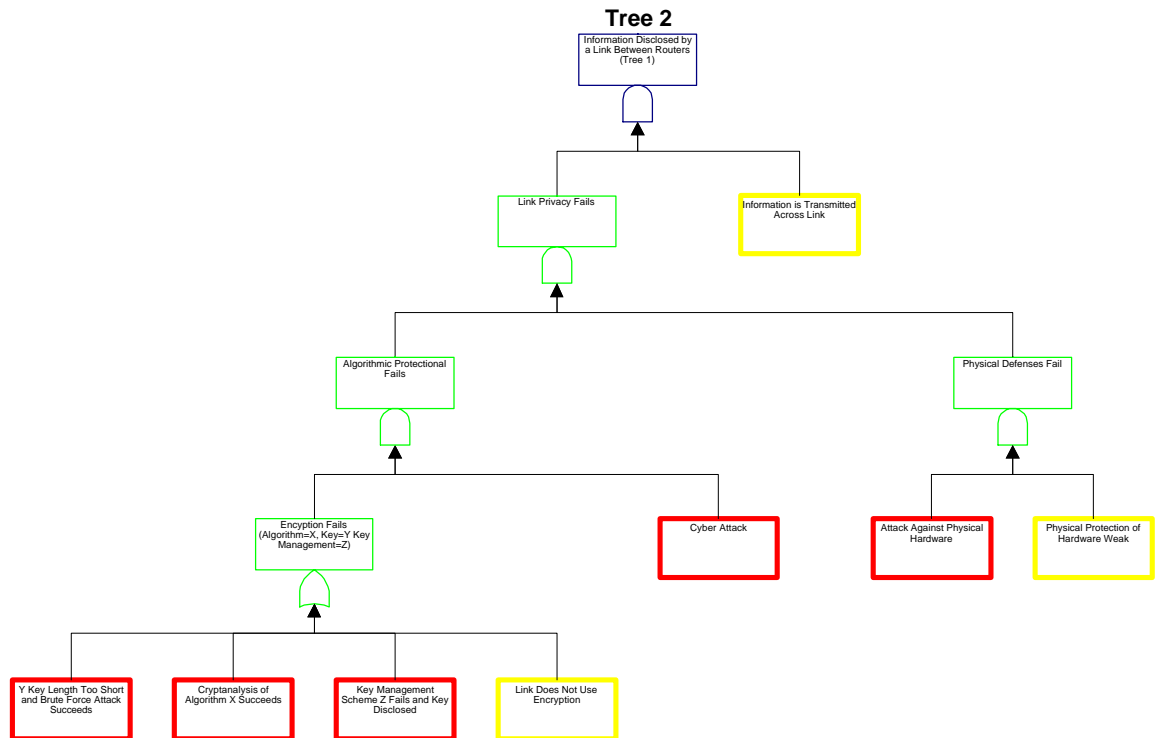
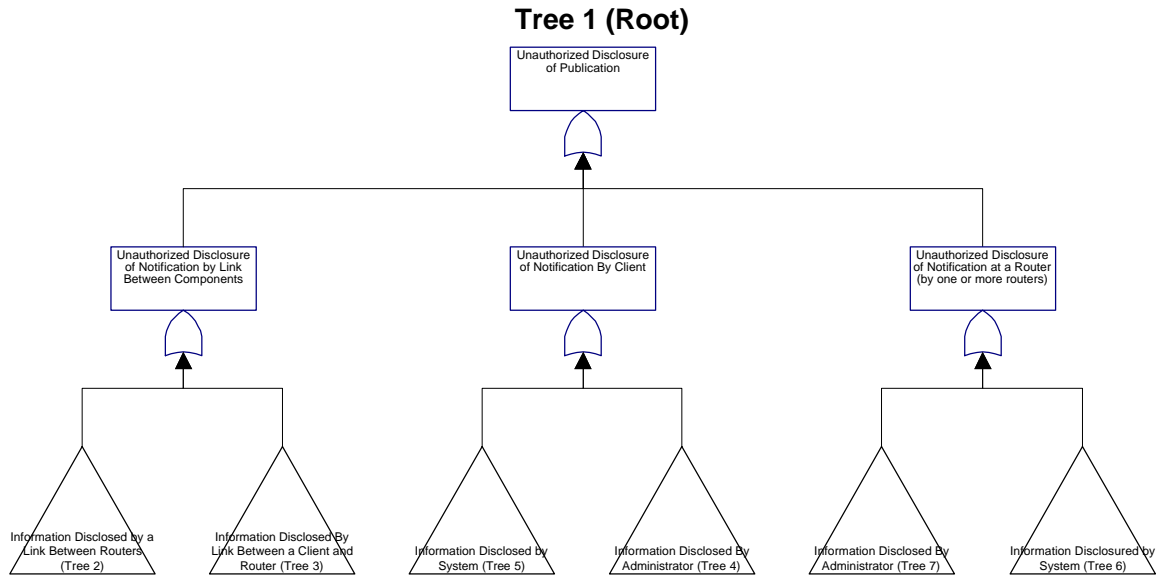
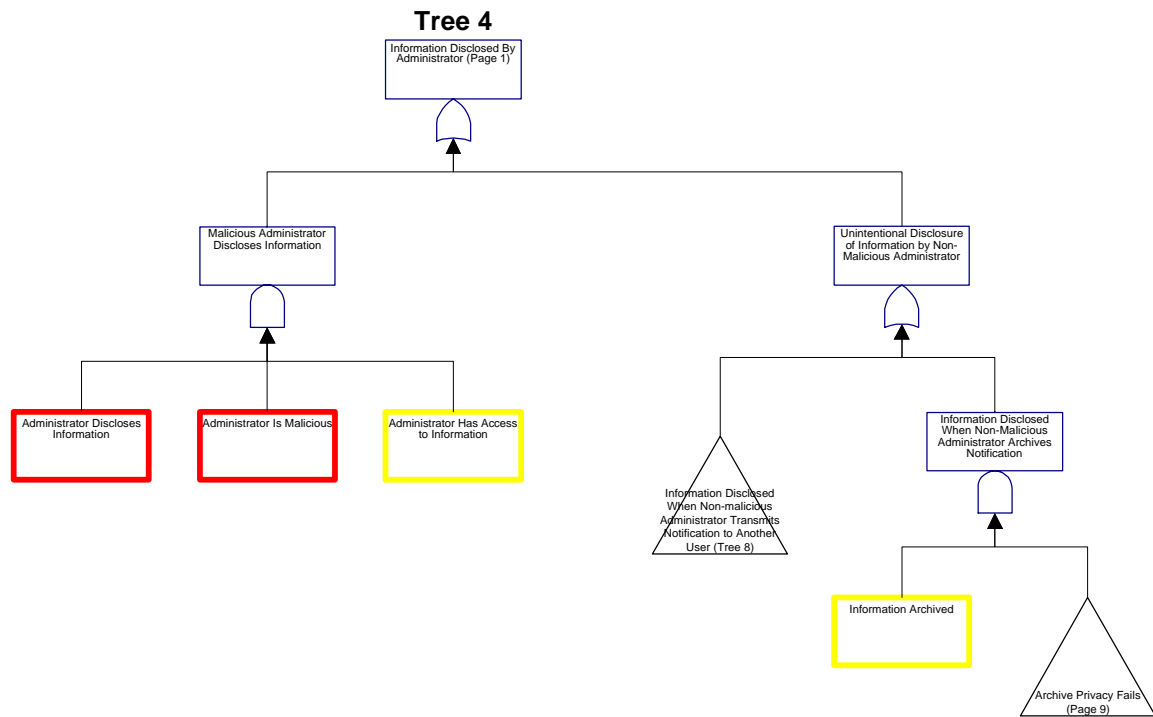
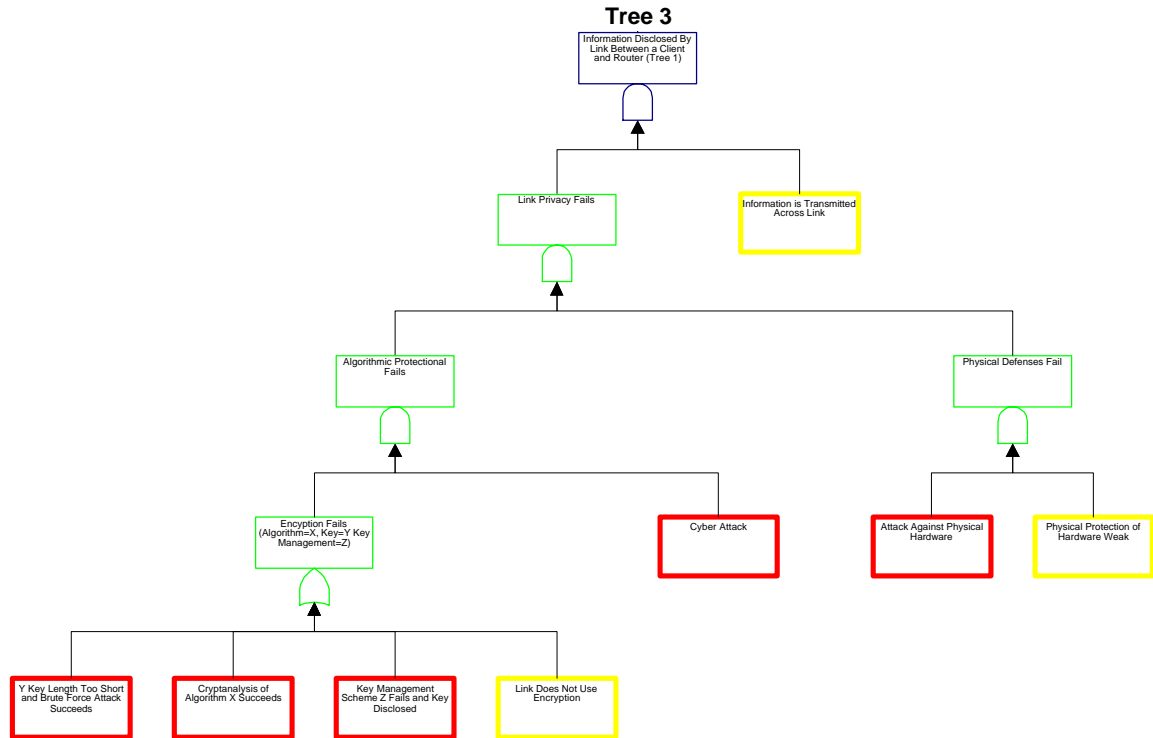


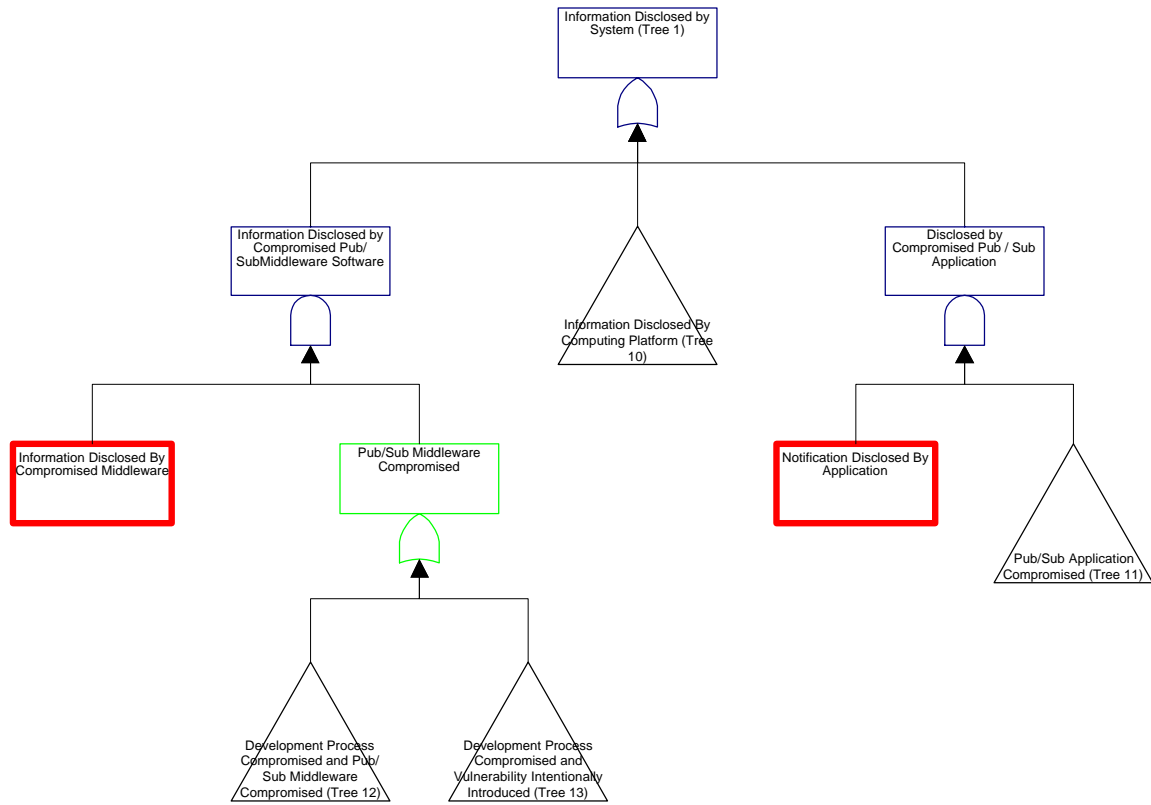
Figure 10. Network Unavailable Fault Tree

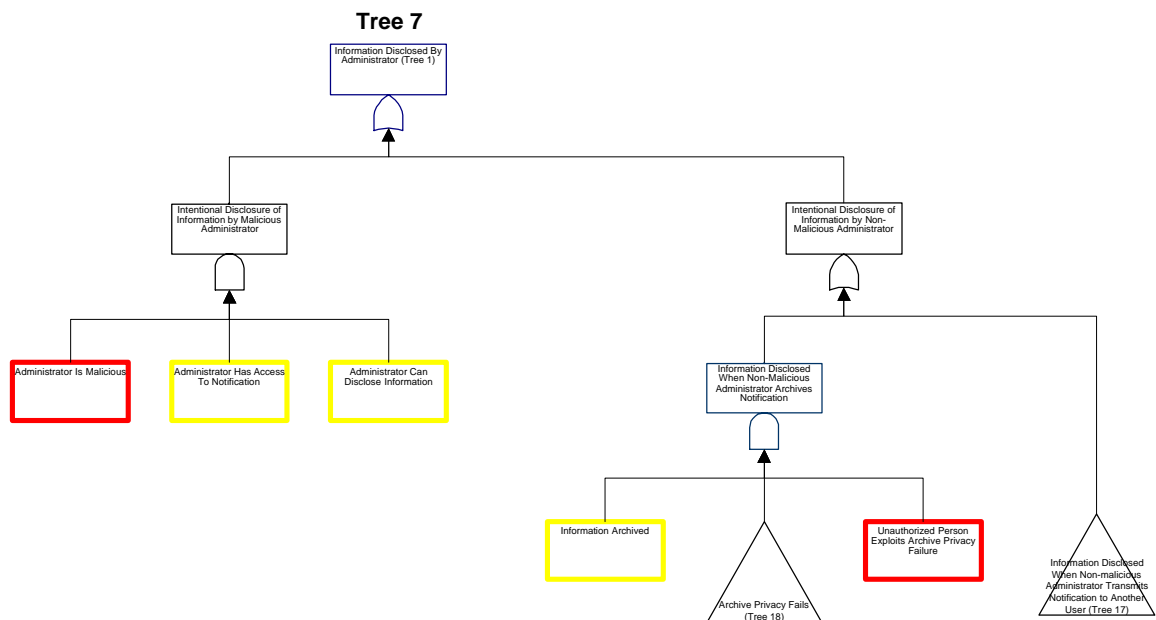
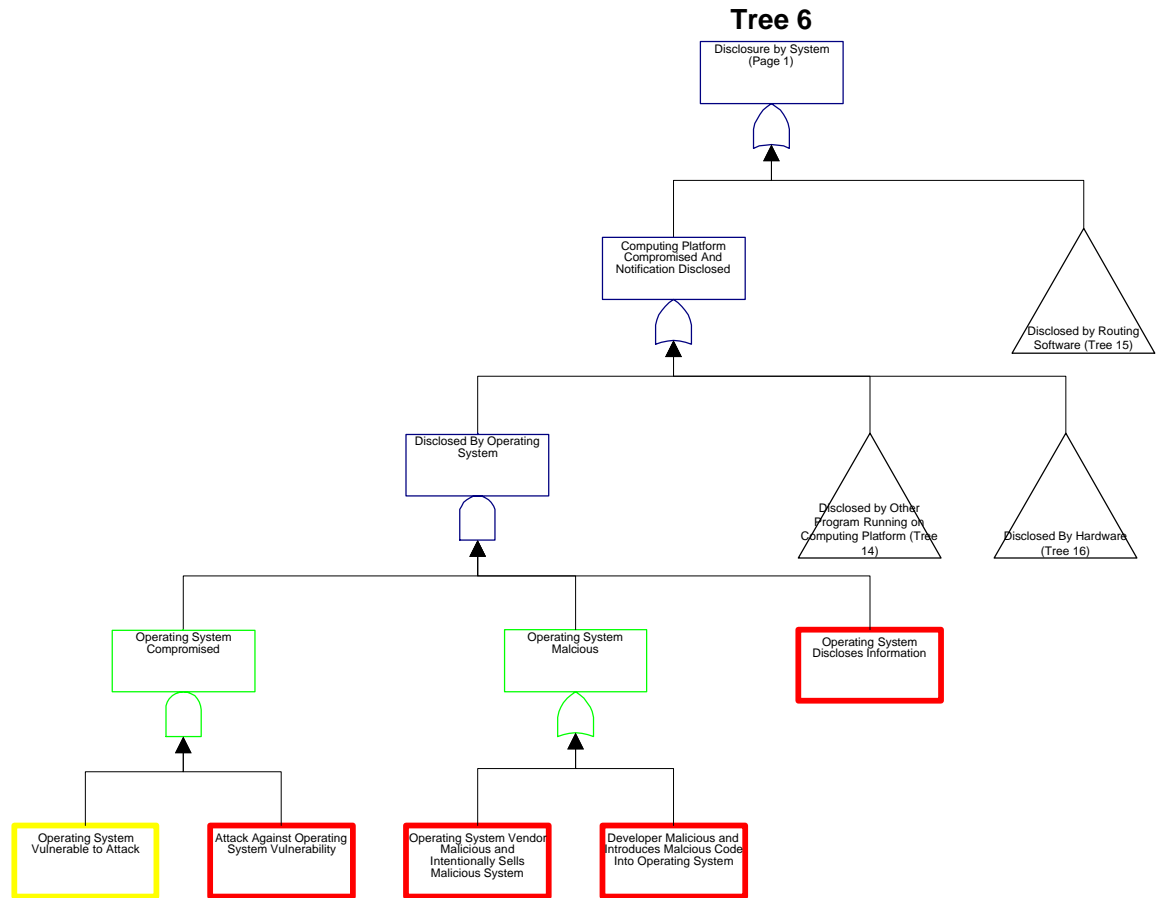
II-5.2 Security Enhanced Fault Tree For Hazard 1

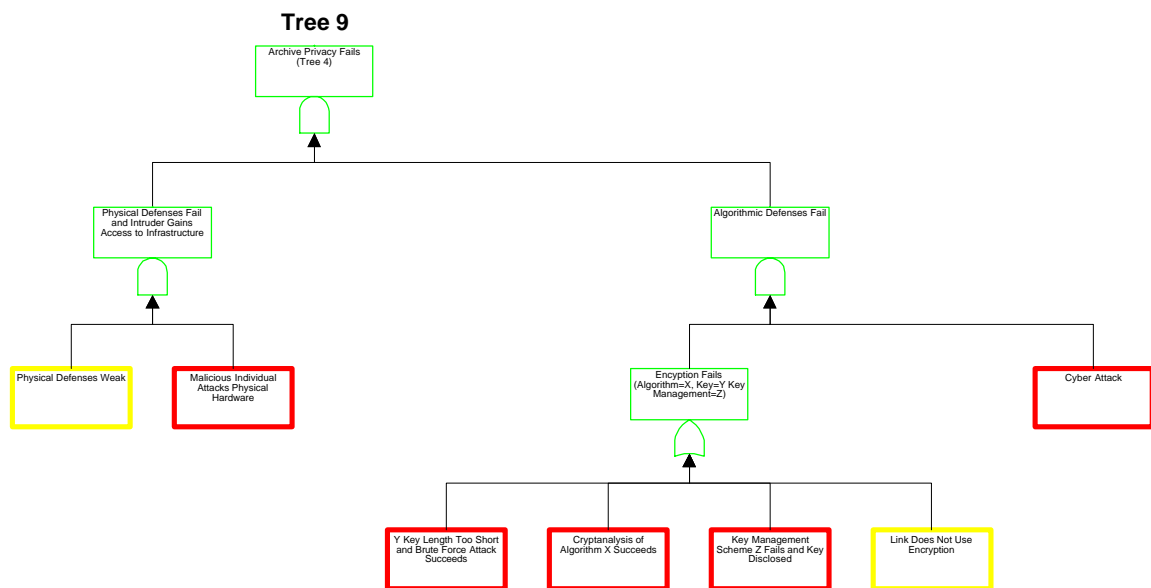
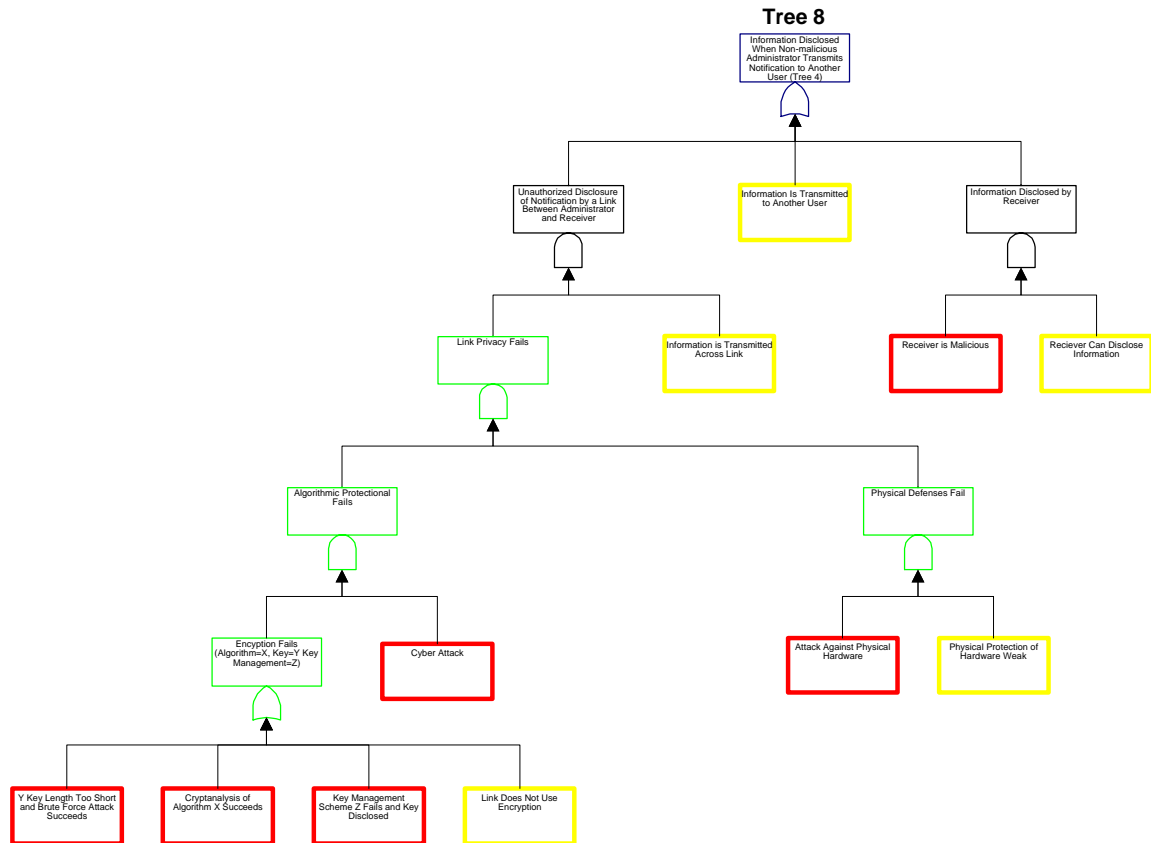


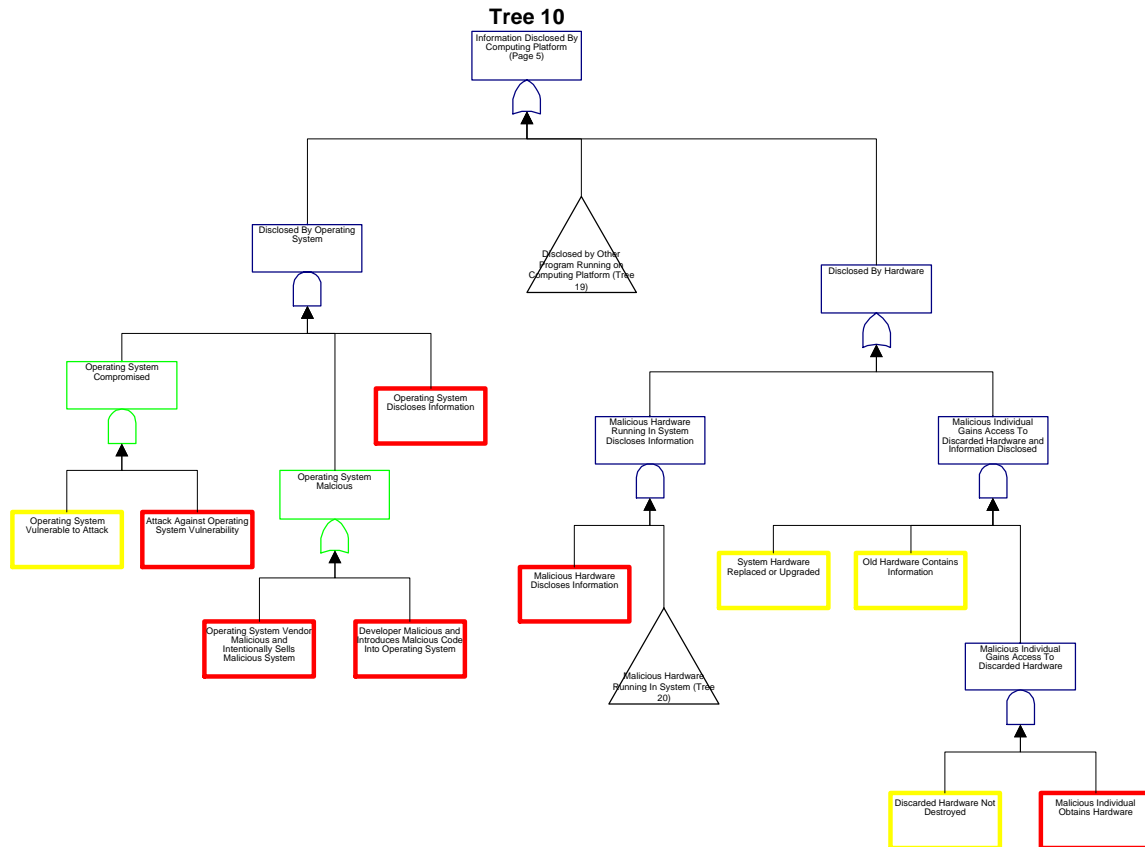


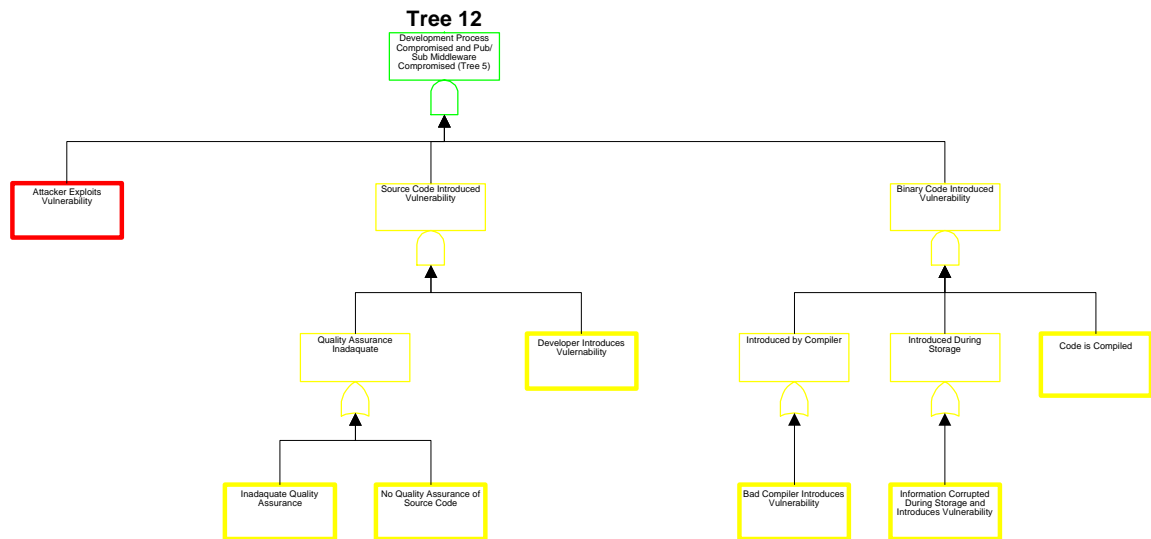
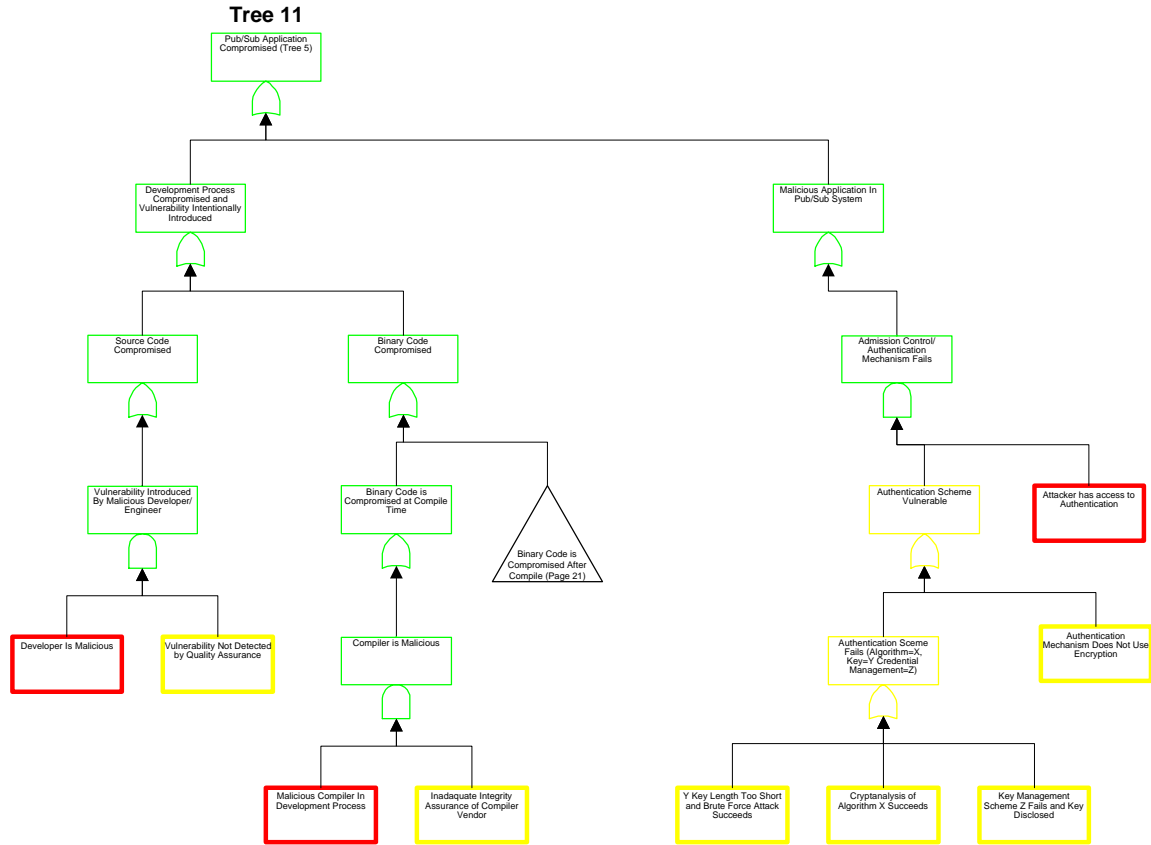
Tree 5



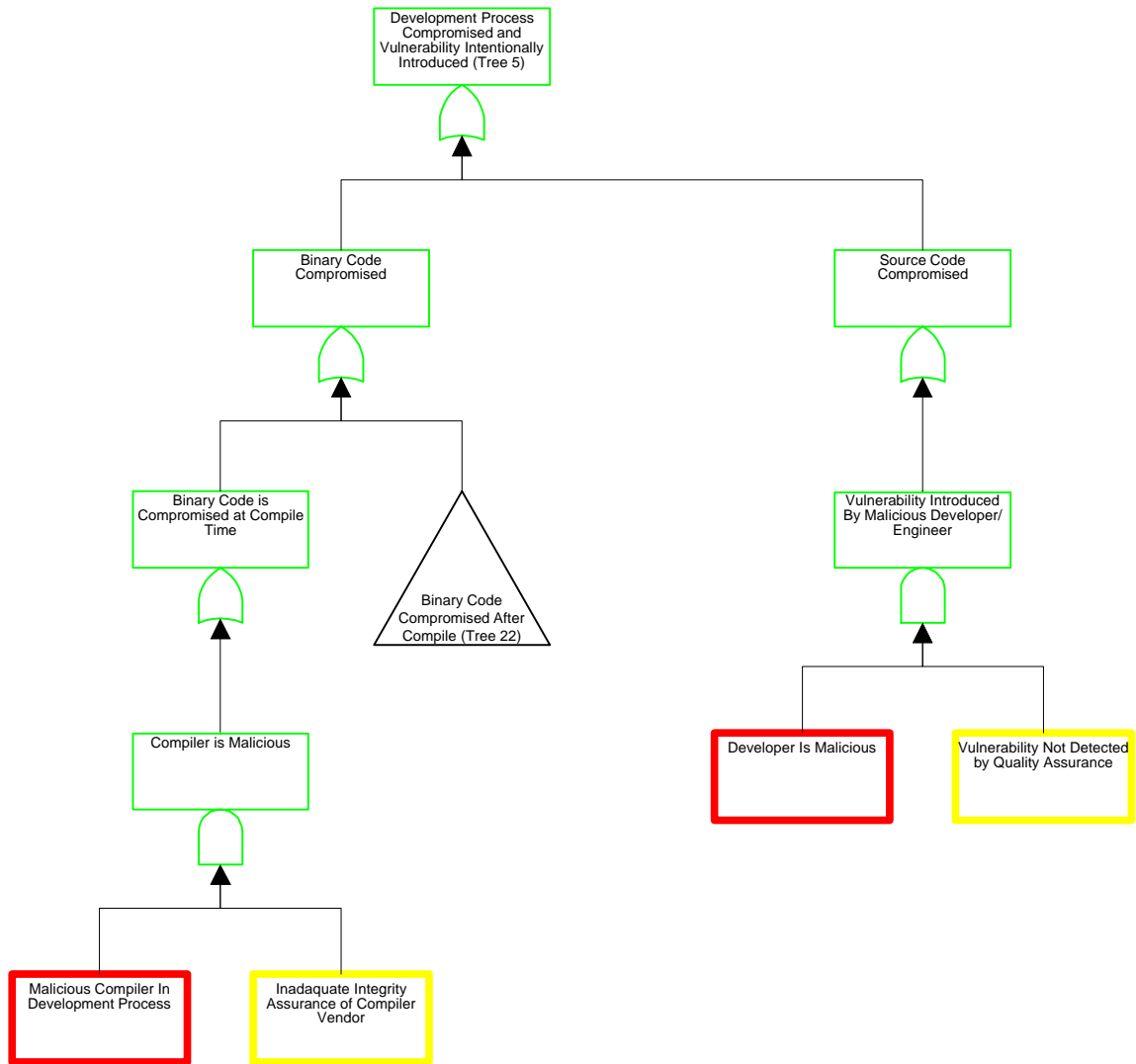


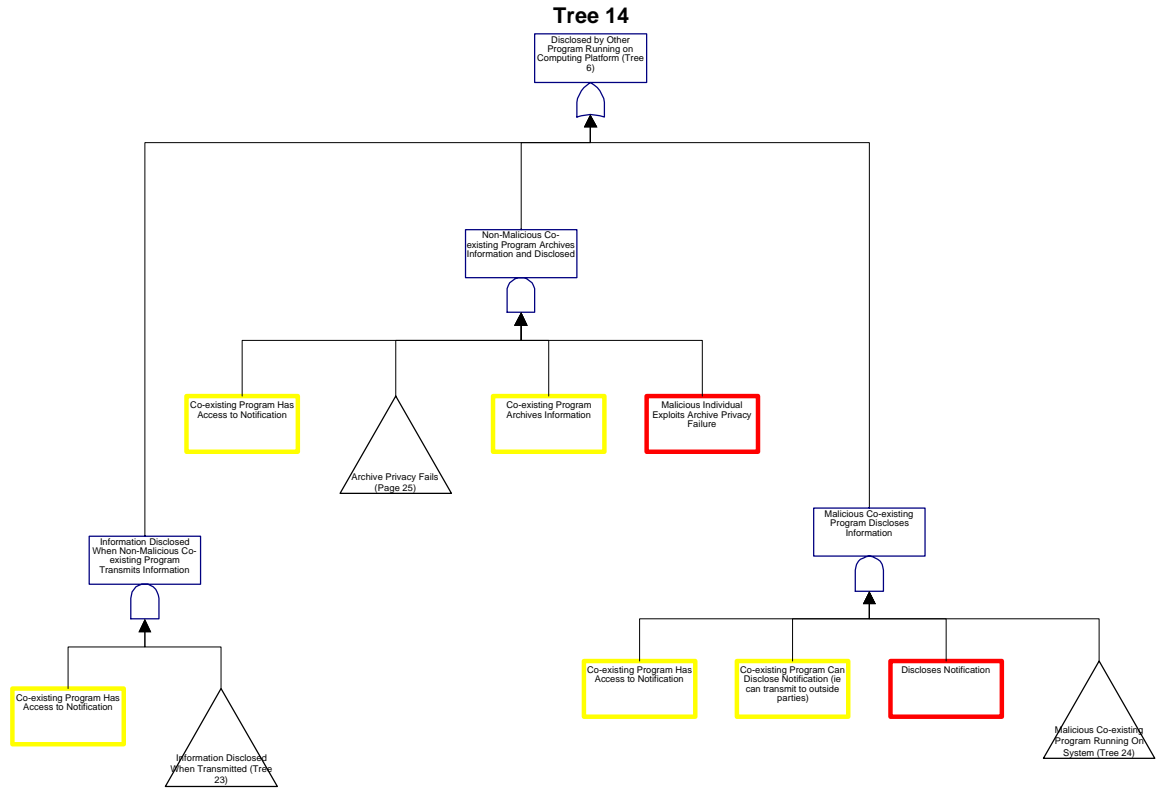


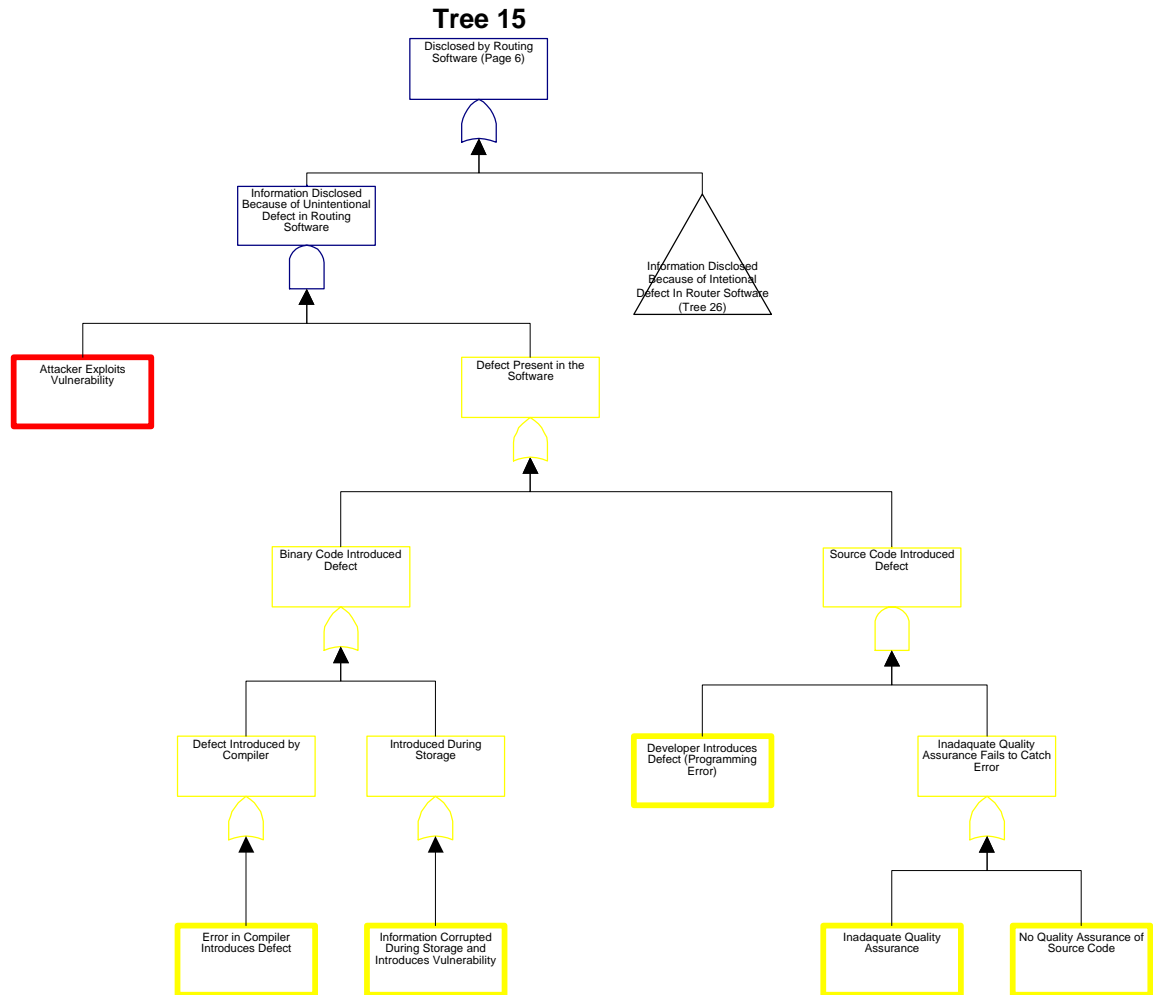


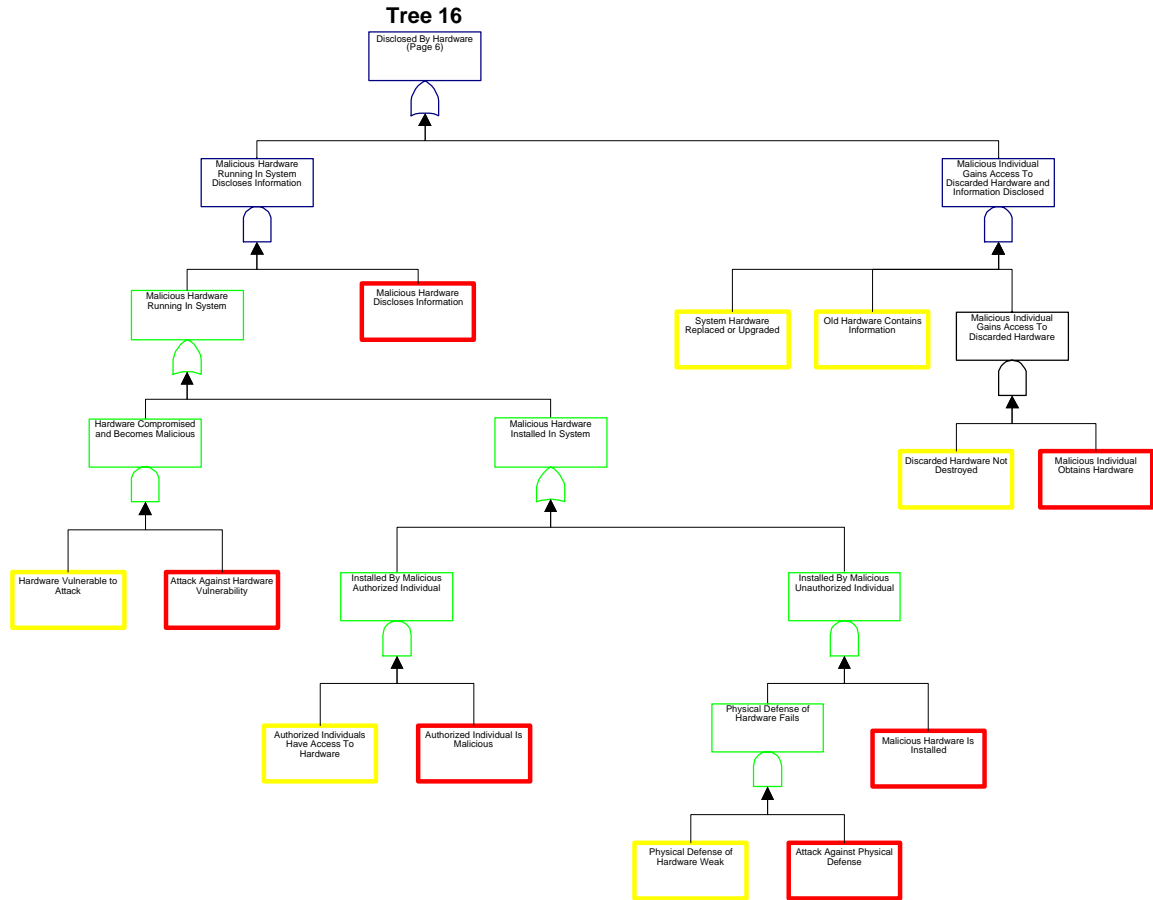


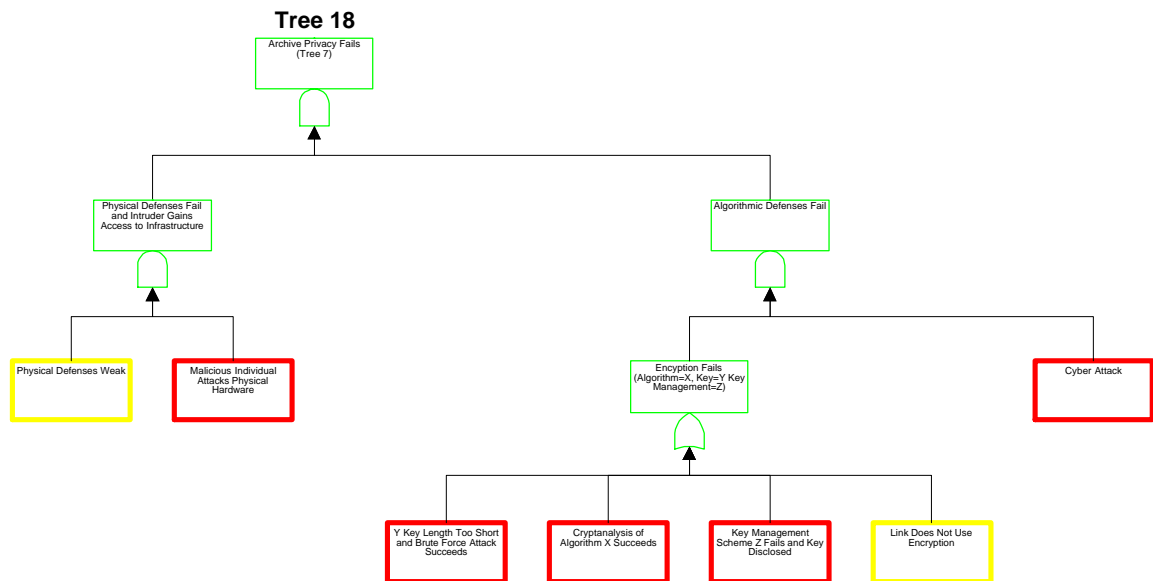
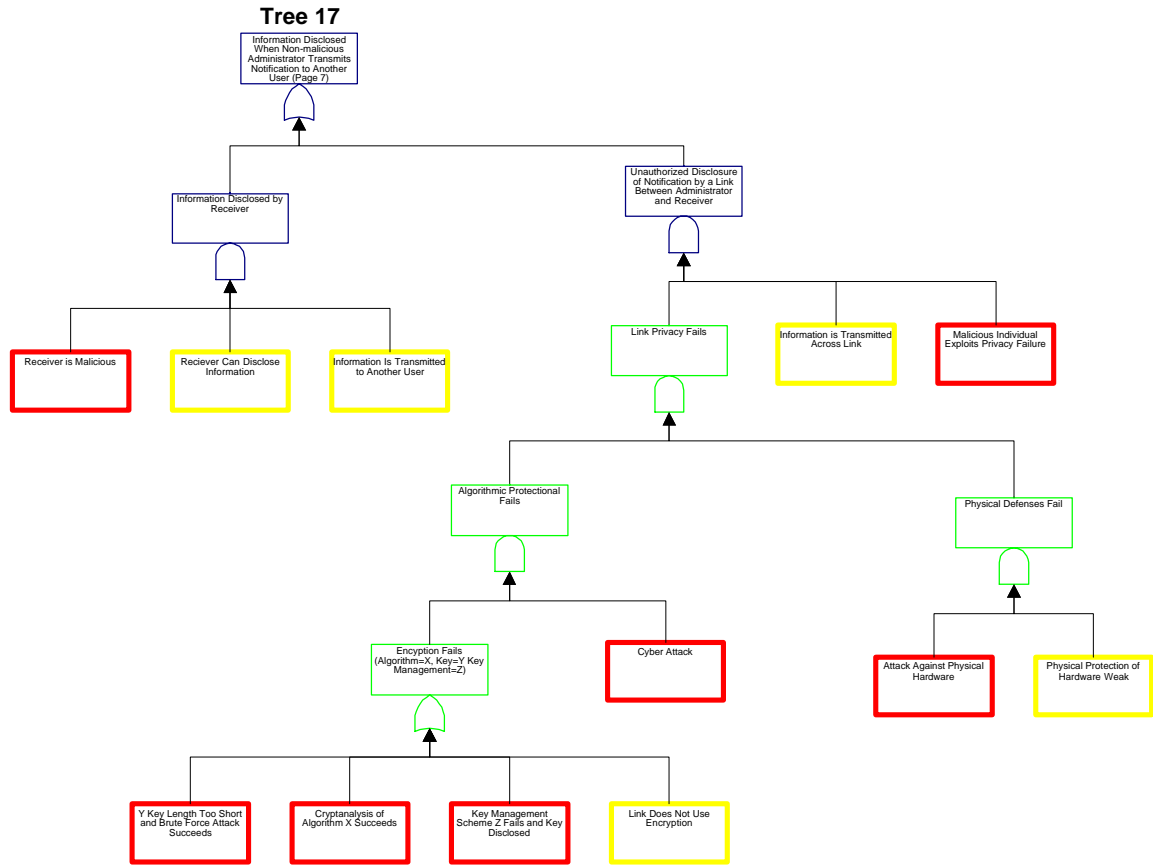
Tree 13

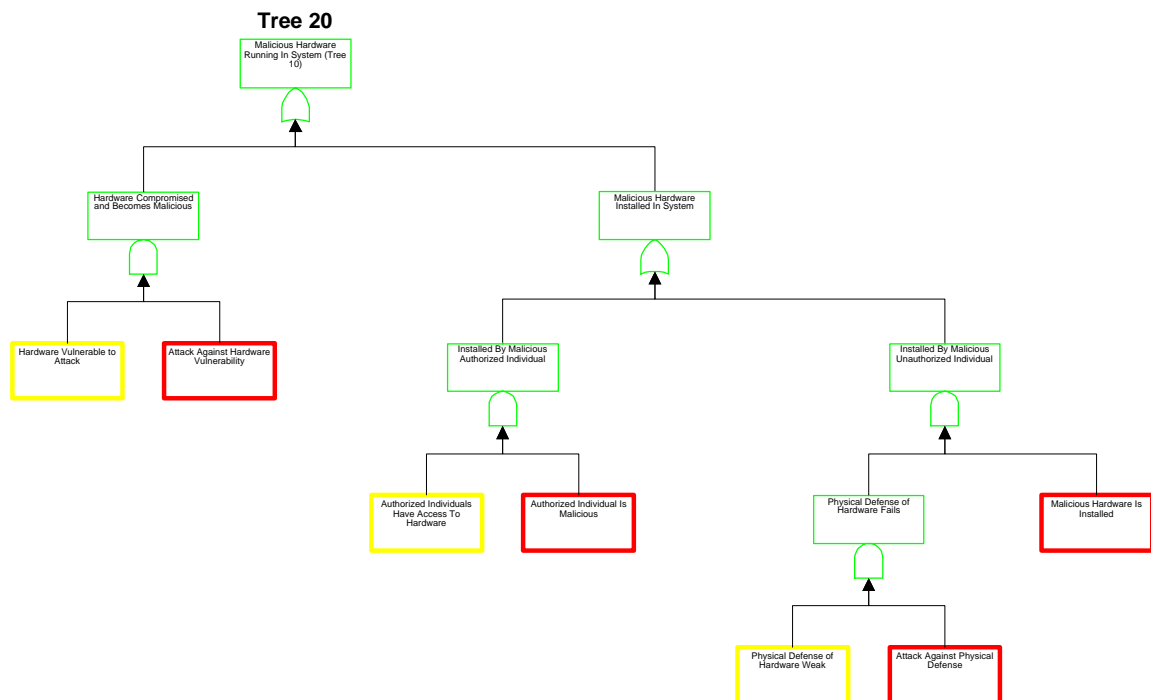
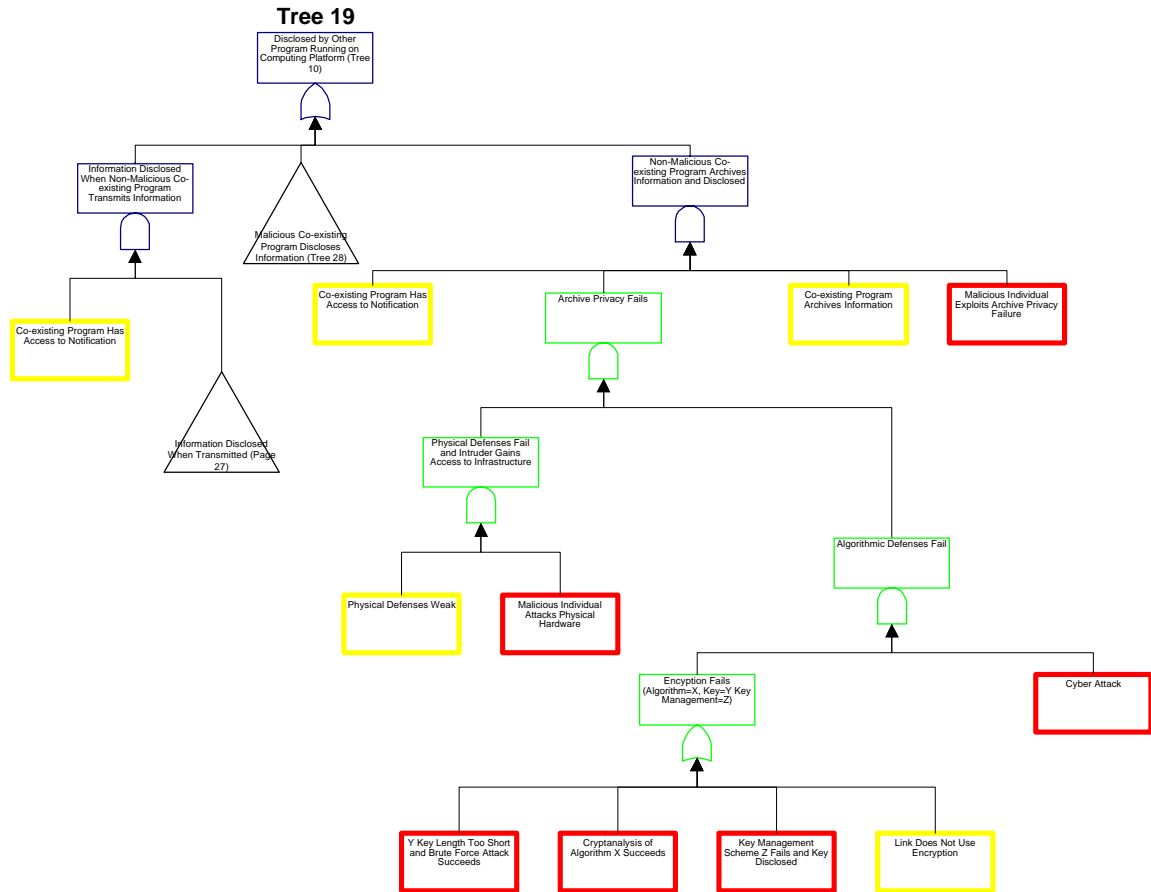


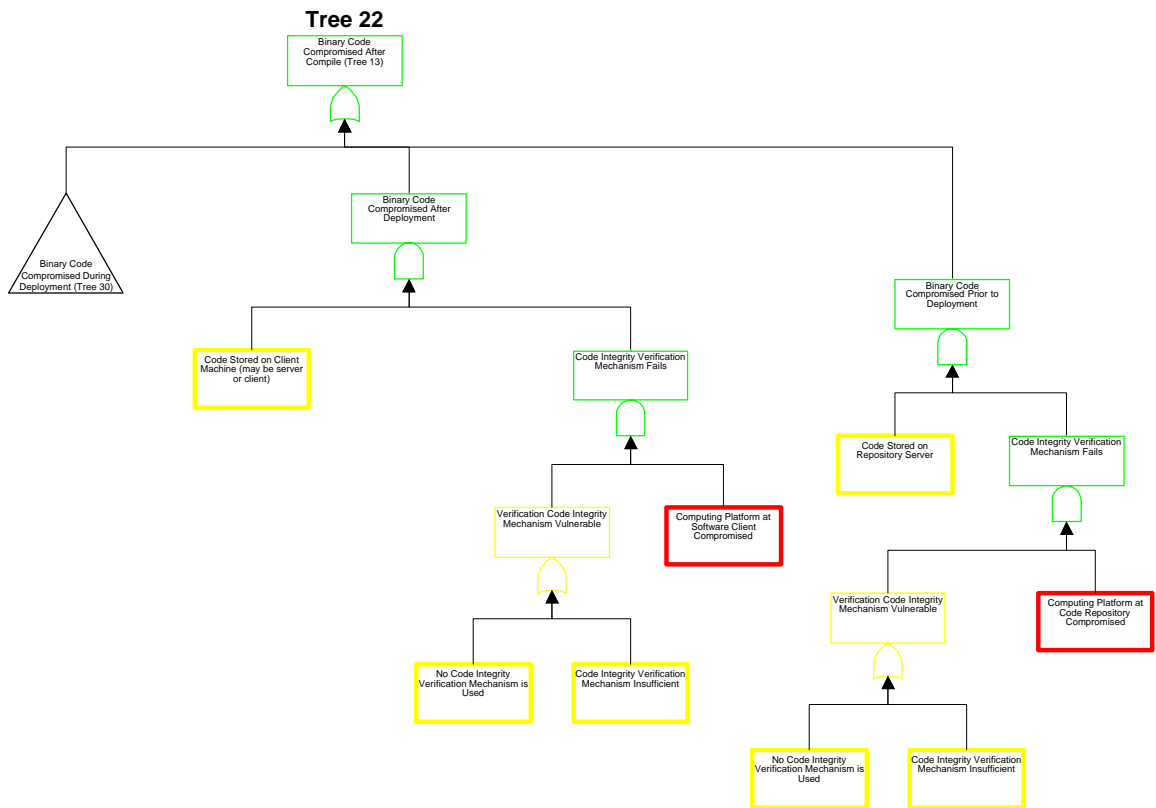
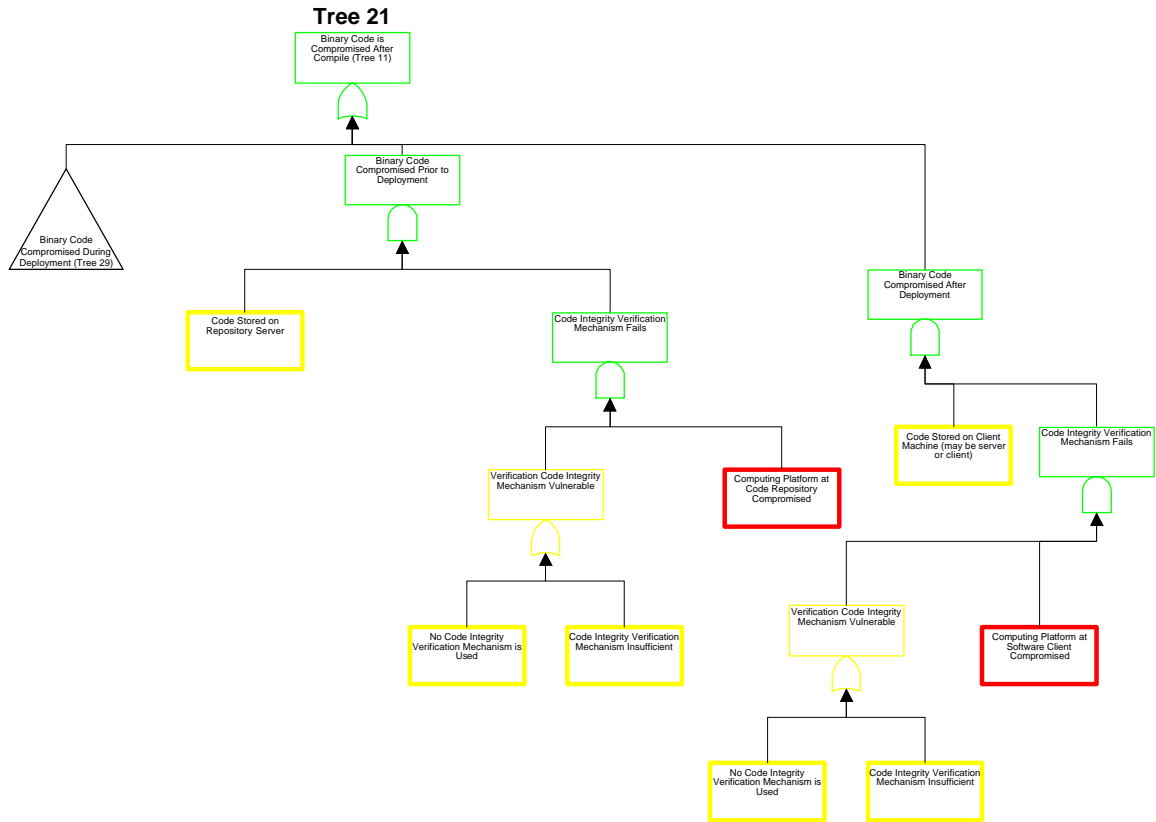


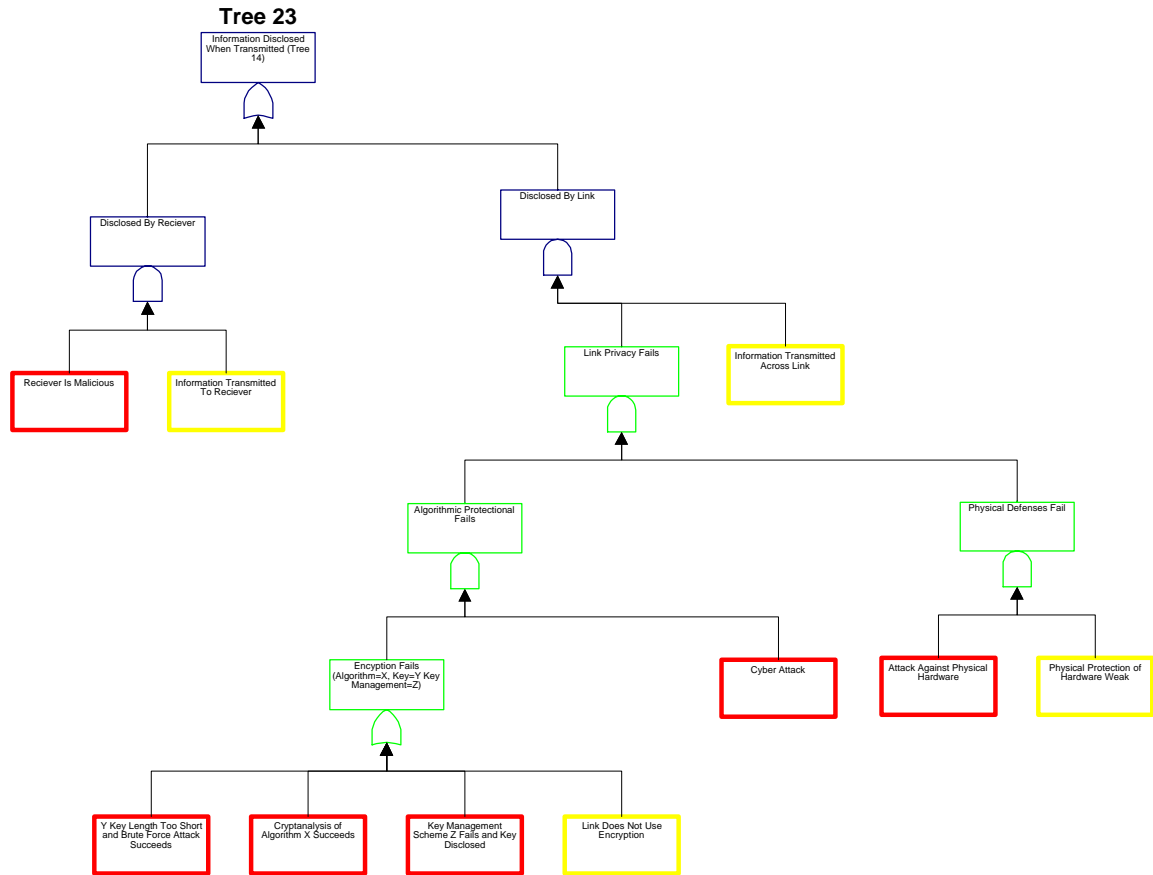


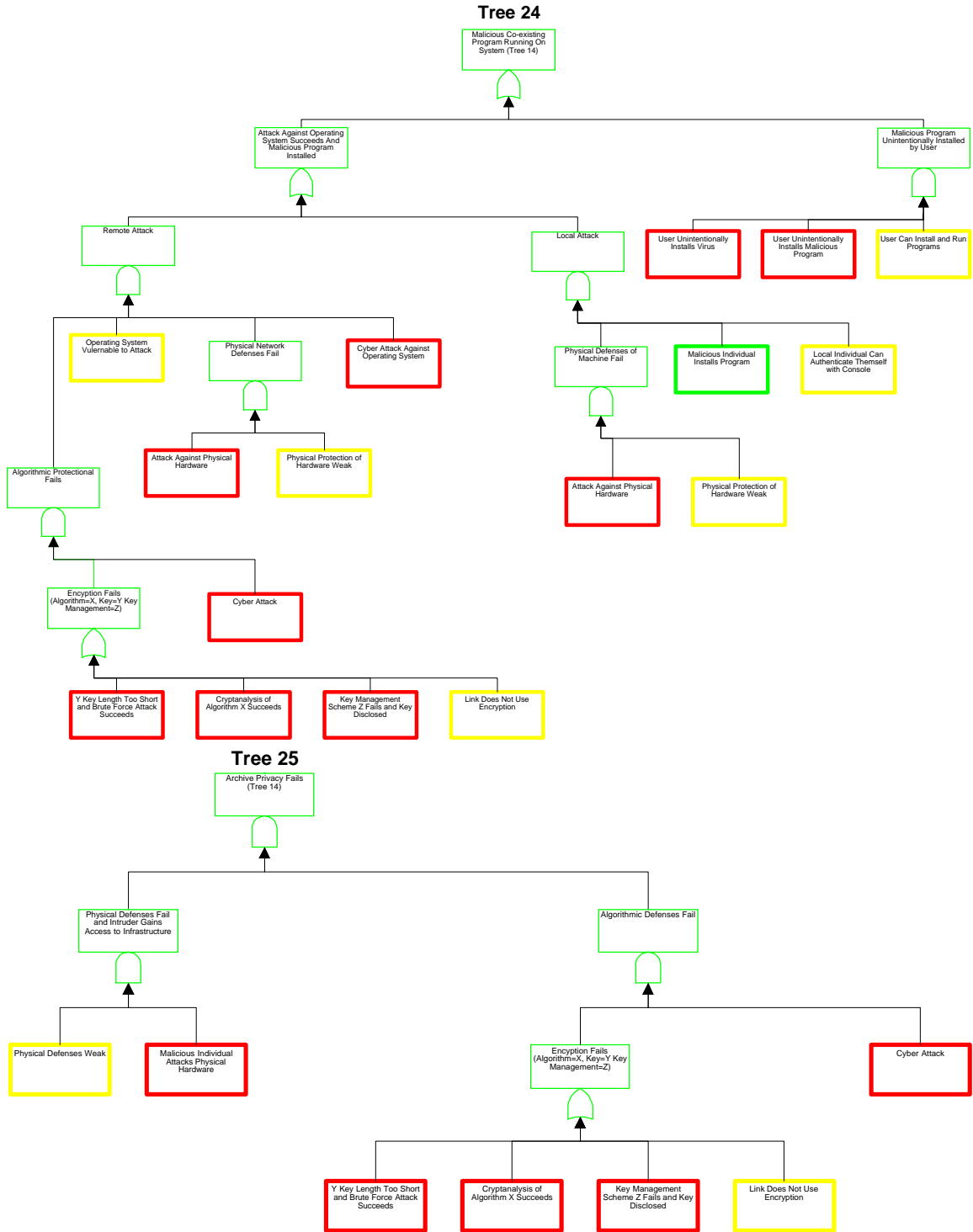




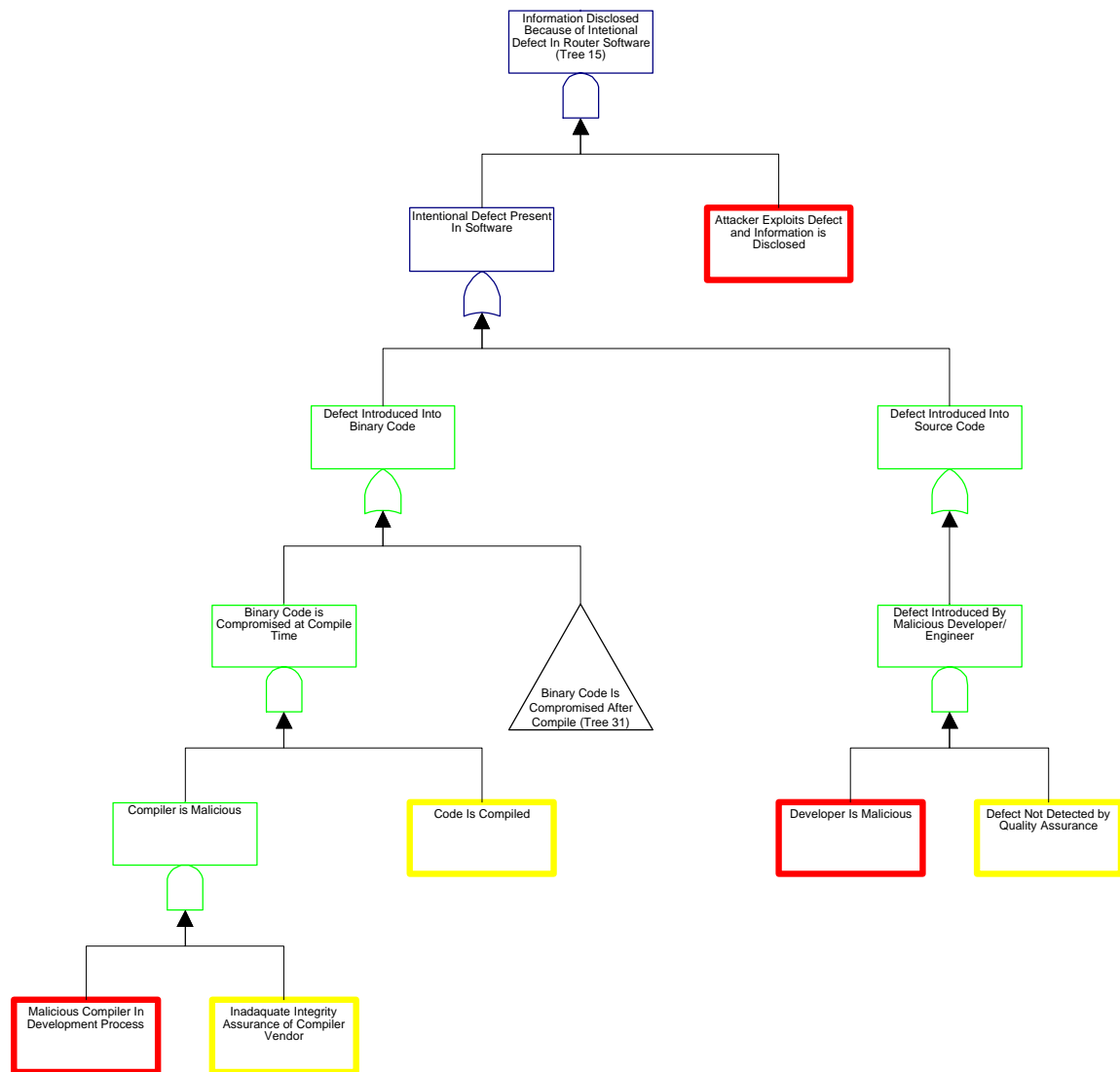


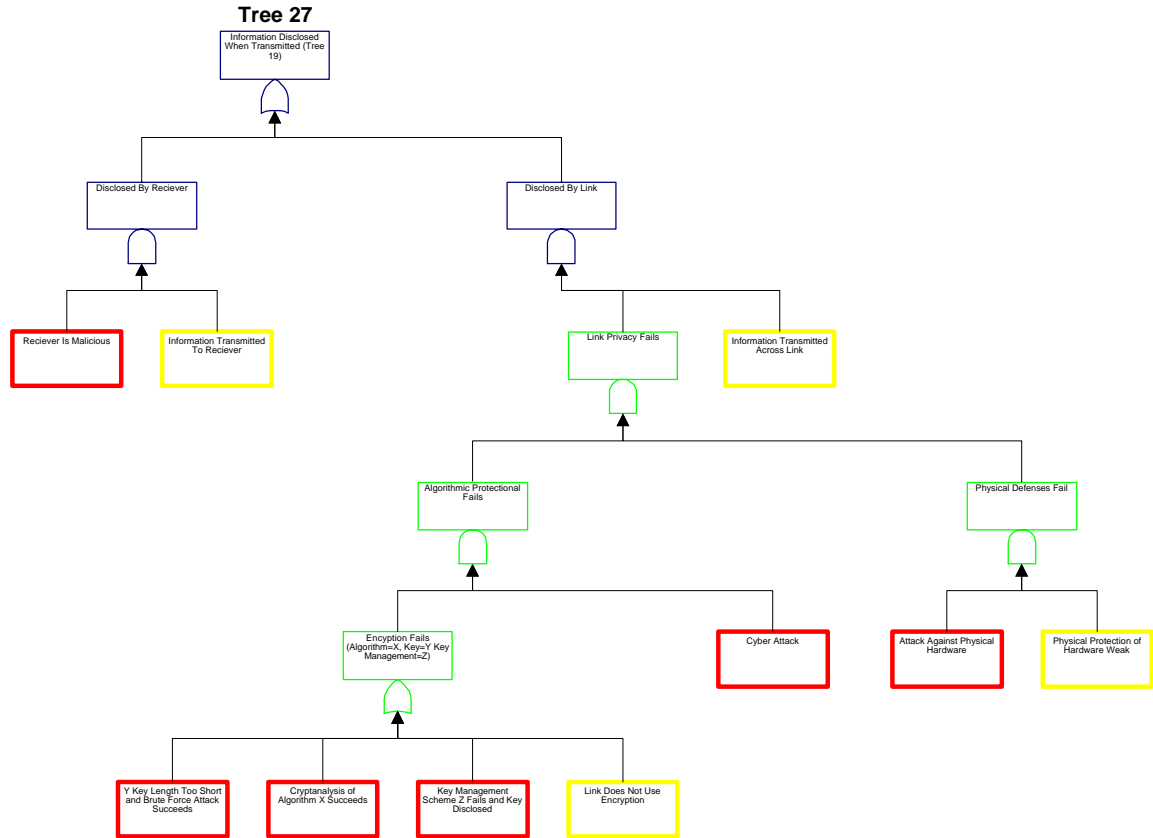


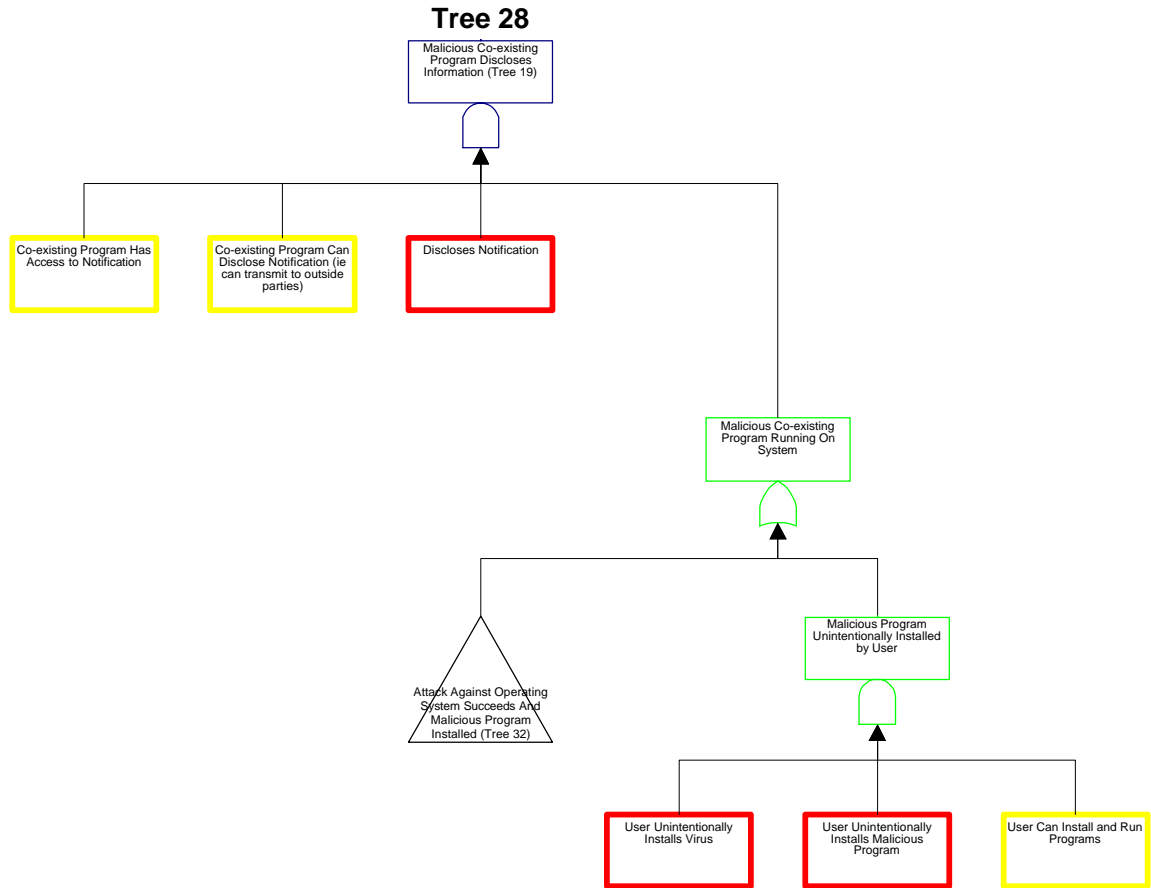


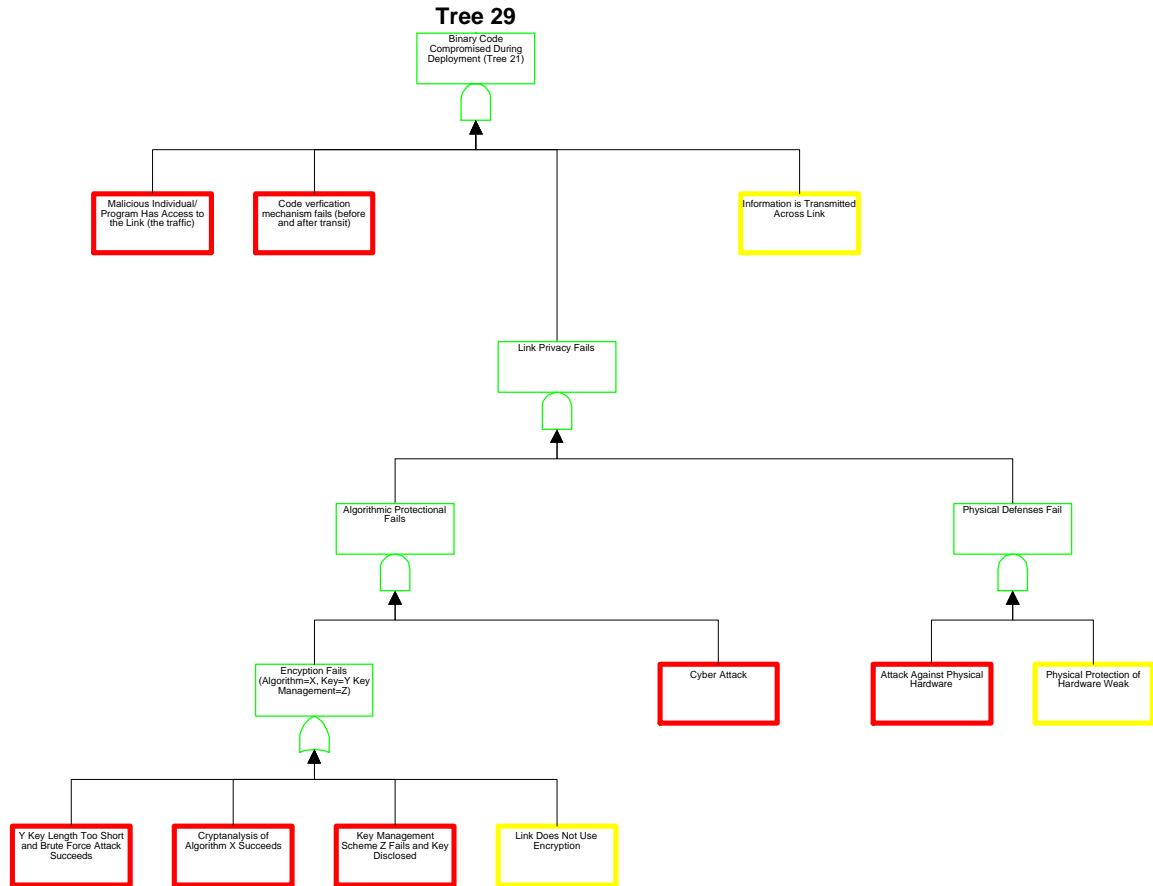


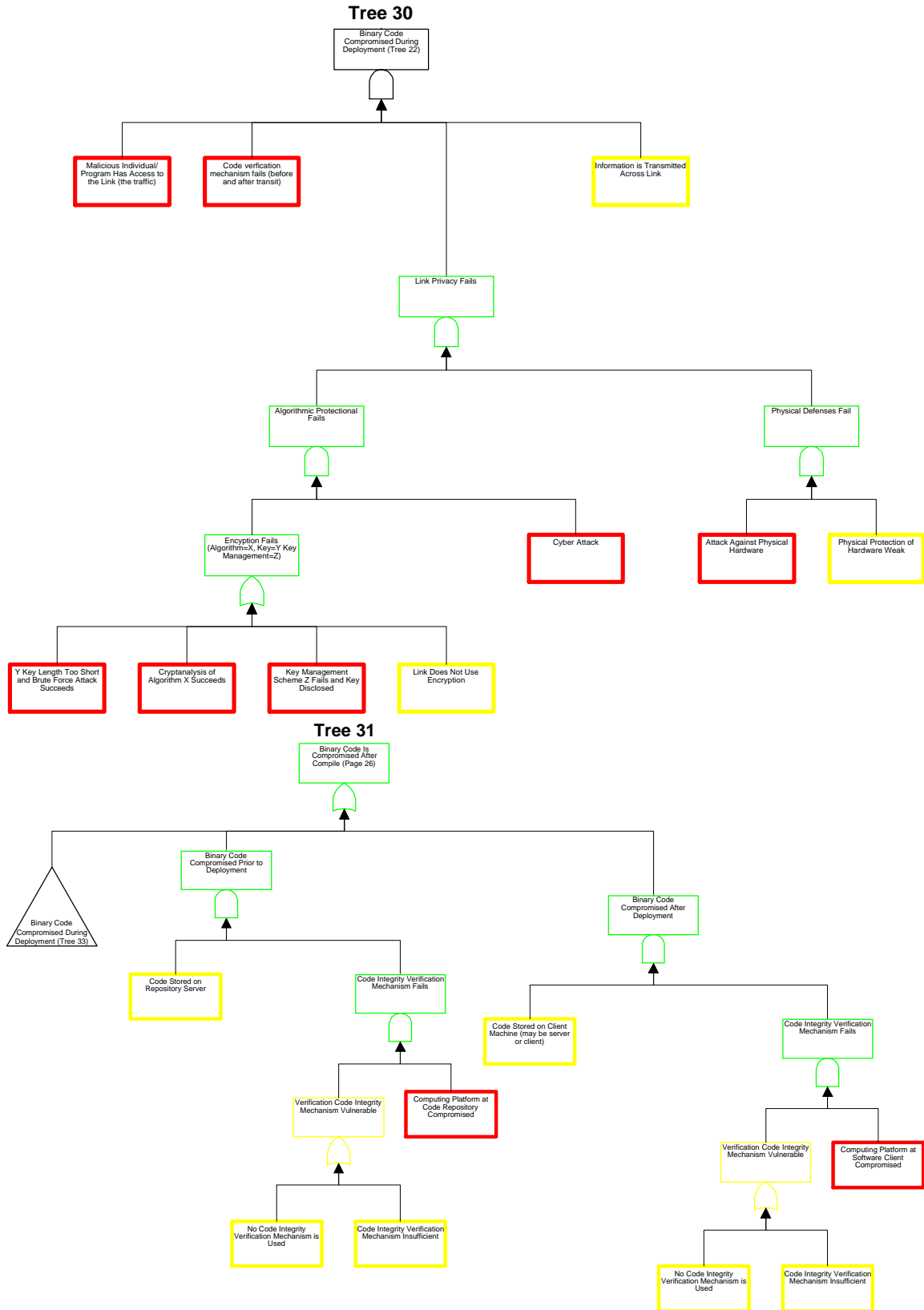
Tree 26

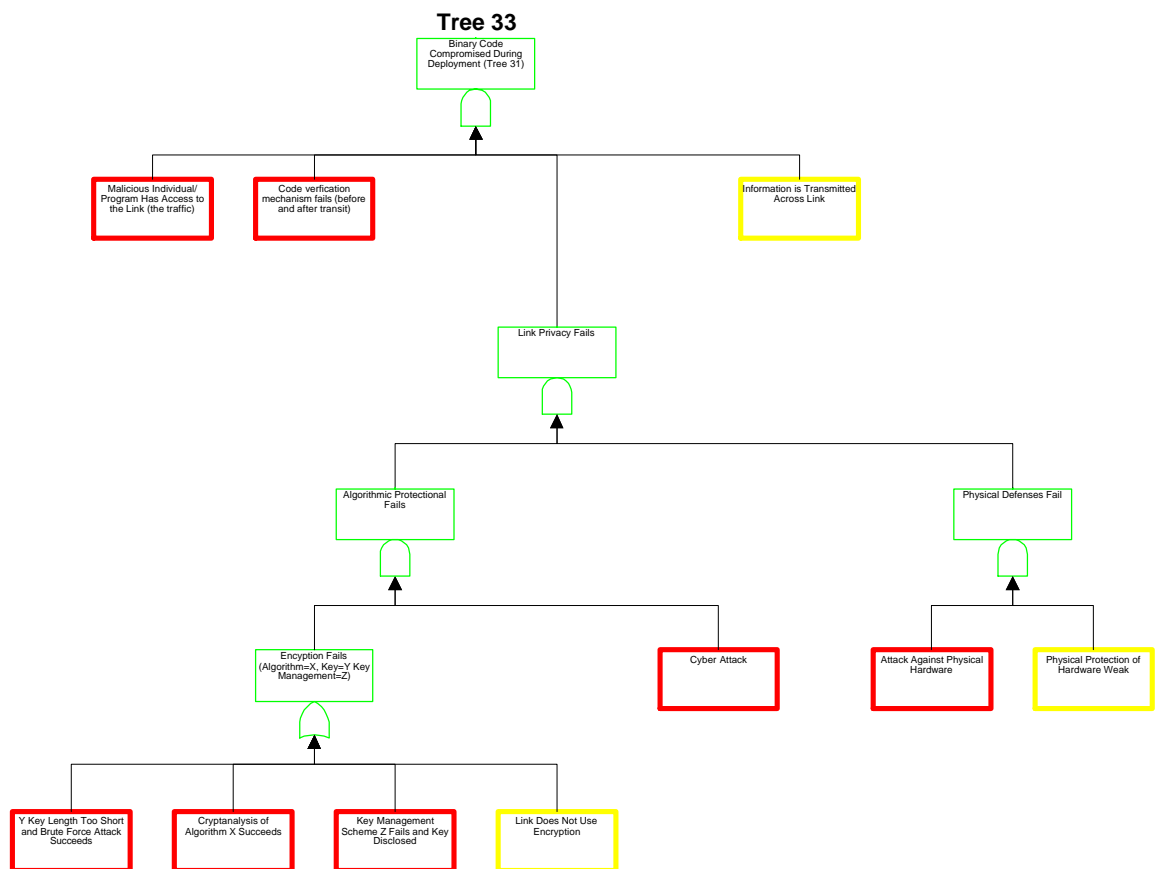
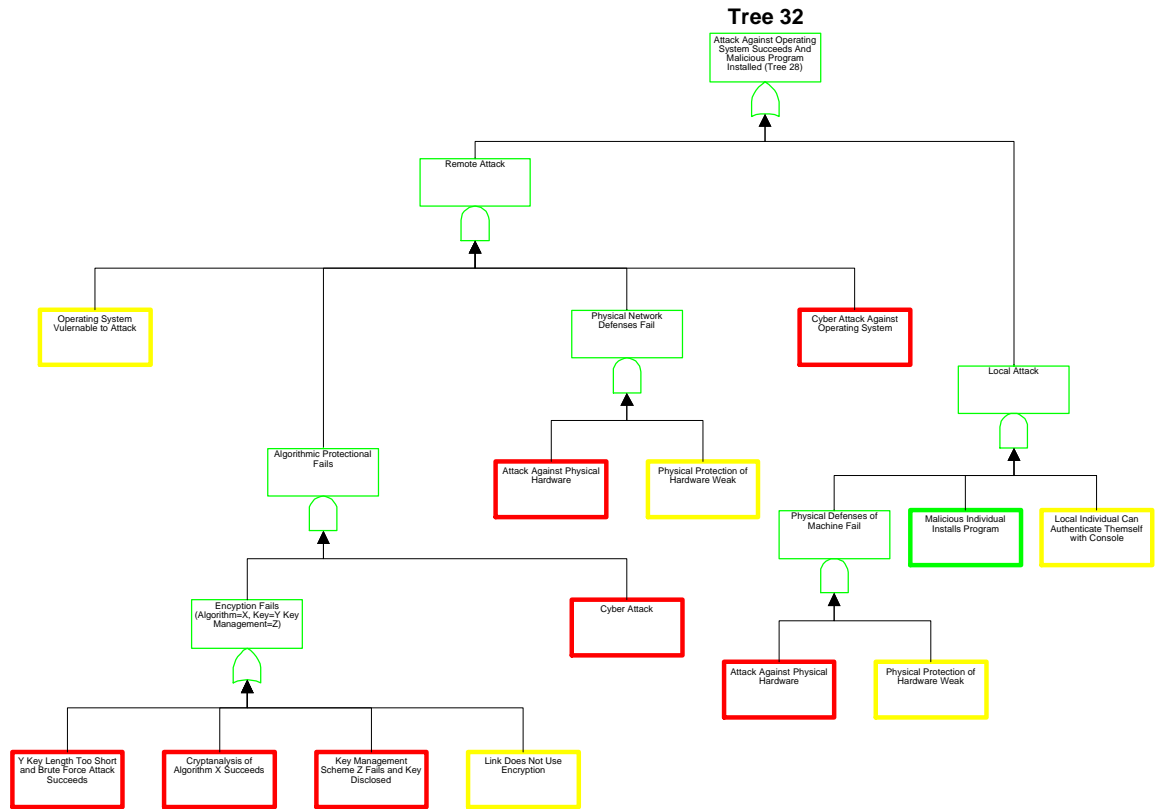












II-6. Conclusions

Information hazard analysis of a typical JBI revealed a number of hazards that must be dealt with for any system that is to be made operational. Subsequent development of a security-enhanced fault tree for a specific security hazard revealed that the hypothesized comprehensive analysis and documentation is precisely what occurred.

The introduction of the four types of fault-tree node provided two major benefits:

- During construction of the fault tree, the need to define the type of each node forced: (1) careful consideration of the event itself; and (2) consideration of whether other related events of different types could have been overlooked.
- Review of the tree both during and after development was undertaken with the colors of the nodes providing insights into the general form of the tree. The complete set of places where the system could be attacked are illustrated by the set of red nodes. The complete set of events that could expose a vulnerability are illustrated by the yellow nodes. And the green nodes representing system compromise show the extent of the tree for which attacks can take place. Being able to see these four sets of nodes is extremely useful in determining the overall state of the system.

II-7. References

- [1] Anderson, R., *Security Engineering: A Guide to Building Dependable Distributed Systems*, Wiley (2001).
- [2] Avizienis, Laprie, Randell, "Fundamental Concepts in System Dependability", IARP / IEEE-RAS Workshop on Robot Dependability, May 2001
- [3] Butler, R. and G. Finelli, "The infeasibility of experimental quantification of life-critical software reliability." IEEE Transactions on Software Engineering, 19(1):3--12, Jan. 1993
- [4] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, "Design and Evaluation of a Wide-Area Event Notification Service". ACM Transactions on Computer Systems, 19(3):332-383 (August 2001).
- [5] Federal Aviation Administration. System Safety Handbook, "Chapter 3: Principles of System Safety," December 30, 2000
- [6] Jha S. and Wing J. "Survivability Analysis of Networked Systems". In Proceedings of the International Conference on Software Engineering, May 2001.
- [7] Jha, S. Oleg, S., Wing, J "Minimization and Reliability of Attack Graphs," IEEE Symposium on Security and Privacy, Oakland, May 2002.
- [8] Kienzle, D. M., and Wulf, W. A., "A Practical Approach to Security Assessment," Proceedings of the 1997 New Security Paradigms Workshop, England, September 1997.
- [9] Krsul, Ivan. "Computer Vulnerability Analysis Thesis Proposal," Technical Report CSD-TR-97-0926, Department of Computer Science Purdue University, April 15, 1997
- [10] Leveson, N., *Safeware: System Safety and Computers*, Addison Wesley (1995).
- [11] Linger, R., Moore, A. "Foundations for Survivable System Development: Service Traces, Intrusion Traces and Evaluations Models" Technical Report CMU/SEI-2001-TR-029, October 2001
- [12] Modarres, M., *What Every Engineer Should Know About Reliability and Risk Analysis*, Dekker (1993).
- [13] Moore, Andrew, Ellison, Robert, Linger, Richard. "Attack Modeling for Information Security and Survivability. Technical Note CMU/SEI-2001-TN001, March, 2001
- [14] Moskowitz, I.S, and M. H. Kang, "An Insecurity Flow Model", Proceedings of the Sixth New Security Paradigms Workshop", Langdale, Cumbria, UK, September, 1997, pp. 61-74.
- [15] Phillips, C. and Swiler, L. A Graph-based System for Network Vulnerability Analysis, New Security Paradigms Workshop, pages 71-79, 1998.

- [16] Piszcz, A., Orlans, N., Eyler-Walker, Z., and Moore, D., Engineering Issues for an Adaptive Defense Network. Mitre Technical Report MTR 01W0000103, June, 2001
- [17] Ritchey, R.W and Ammann, P. "Using model checking to analyze network vulnerabilities." In Proceedings of IEEE Symposium on Security and Privacy, pages 156–165, May 2001.
- [18] Sheyner, J. Haines, S. Jha, R. Lippmann, and J.M. Wing. Automated generation and analysis of attack graphs. In Proceedings of IEEE Symposium on Security and Privacy, May 2002.
- [19] Schneier, Bruce. Attack Trees. Dr. Dobbs Journal, December 1999
- [20] Schneier, B., *Secrets and Lies : Digital Security in a Networked World*, Wiley (2000).
- [21] Storey, N., *Safety-Critical Computer Systems*, Addison Wesley (1996).
- [22] United States Department of Commerce National Institute of Standards And Technology. PBX Vulnerability Analysis: Finding Holes in your PBX Before Someone Else Does, Special Publication 800-24

APPENDIX C

“Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems”

Security Issues and Requirements for Internet-Scale Publish-Subscribe Systems

Chenxi Wang, Antonio Carzaniga, David Evans, Alexander L. Wolf

chenxi@cmu.edu, carzanig@cs.colorado.edu, evans@virginia.edu, alw@cs.colorado.edu

Abstract

Publish-subscribe is a communication paradigm that supports dynamic, many-to-many communications in a distributed environment. Content-based pub-sub systems are often implemented on a peer-to-peer infrastructure that enables information dissemination from information producers (publishers) to consumers (subscribers) through a subscription mechanism. In a wide-area pub-sub network, the pub-sub service must handle information dissemination across distinct authoritative domains, heterogeneous platforms and a large, dynamic population of publishers and subscribers. Such an environment raises serious security concerns. In this paper, we investigate the security issues and requirements that arise in an internet-scale content-based pub-sub system. We distinguish among those requirements that can be achieved with current technology and those that require innovative solutions.

1 Introduction

Today's mission-critical systems make extensive use of distributed computing over large, heterogeneous networks. A promising technology in achieving distributed computing is the use of publish-subscribe mechanisms (hereafter refer to as pub-sub). A pub-sub system is a communication infrastructure that enables data access and sharing over disparate systems and among inconsistent data models [7]. Gnutella [16] is an example of a pub-sub system. This paper focuses on content-based publish-subscribe where subscribers register interest to information and the infrastructure routes the information to the subscribers based on the information content and the user subscriptions.

In a wide-area content-based pub-sub network, the underlying pub-sub infrastructure is often implemented as a collection of network servers communicating with each other in a peer-to-peer fashion [8]. In such an environment, the pub-sub service must handle information dissemination across distinct authoritative domains, heterogeneous platforms and a large, dynamic population of publishers and subscribers. Many security concerns exist in such an environment. For example, delivering information to interested (and authorized) parties only is an information privacy concern, and so is the concern of keeping the subscription information private. Additionally, the integrity and availability of the pub-sub mechanism must be ensured.

The current designs of pub-sub systems tend to focus on the performance, scalability and

expressiveness issues of the mechanism [8][24]. In this paper, we investigate the basic security issues for pub-sub systems. We will distinguish between those that can be achieved with current technology and those that merit innovative solutions. We will not, however, attempt to design a security model in this paper.

2 Publish-subscribe systems

A pub-sub system is a routing network that delivers datagrams from publishers to interested subscribers. Unlike multicast group communications where group addresses and memberships are statically bound, pub-sub systems use a communication model where the eligibility of group membership is evaluated dynamically. Such a communication system has many potential benefits. For instance, instead of requiring publishers to identify destination addresses for their messages (potentially requiring multiple messages to multiple destinations), a pub-sub network can handle message routing in a way that avoids unnecessary message replications.

In a content-based pub-sub system, the network supports a language that specifies the publication and subscription interface. For example, a subscriber specifies a subscription function defined over the content of datagrams. A publisher publishes a datagram composed from a set of legal vocabularies in the language. When a publisher sends out a datagram, the network attempts to deliver that datagram to every interface whose subscription function returns true when applied to the content of the datagram.

Communication in a pub-sub system is inherently a multi-party, many-to-many interaction. In this paper, we consider a communication model illustrated by the example in Figure 1. The pub-sub communication protocol can be viewed as follows:

- 1) Subscribers S_1 , S_2 , and S_3 , send subscriptions f_1 , f_2 , and f_3 respectively to some hosts in the network.
- 2) Publisher P_1 sends a datagram d to some entry point of the network, with $f_1(d) \wedge f_2(d) \wedge \neg f_3(d)$; that is, d matches subscriptions f_1 and f_2 but not f_3 .
- 3) The network determines the matching relationships and sends d to S_1 and S_2 .

Note that the publisher only sends d into the network once and need not know anything about the subscribers or subscription functions.

We use two example applications throughout this paper. We believe that they are typical of many of the ways that pub-sub will be used. In the later part of this paper, whenever possible we will discuss the security issues within the context of these examples.

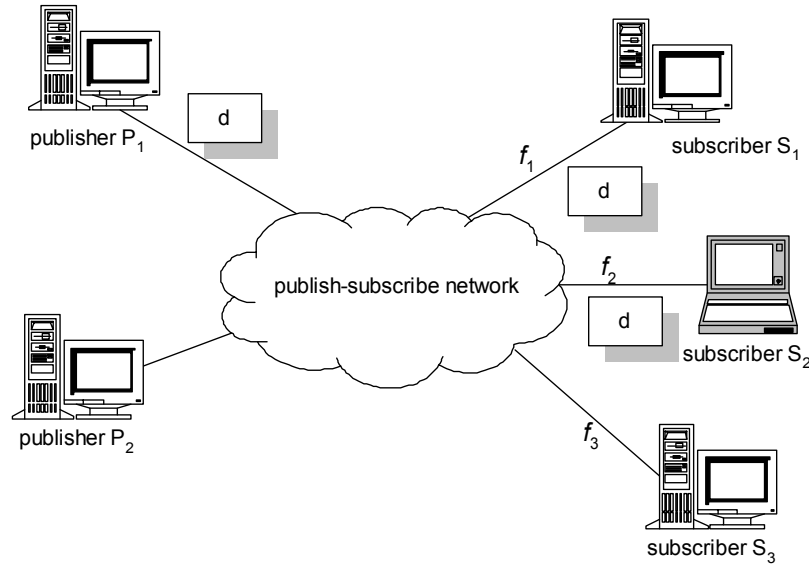


Figure 1: A publish-subscribe system

Stock quotes dissemination: Consider an application that uses a pub-sub system to disseminate real-time stock quotes. Subscribers specify the stock symbol and a schema (e.g., the frequency of quotes) based on which they will receive quotes.

Human resource resume circulation: Consider an application that allows users (publishers) to post their resumes and sells the resume information to interested human resource offices (subscribers). Subscribers specify key words indicating a particular background to search for relevant resumes.

3 Security requirements

The security requirements for a pub-sub system can be divided into the requirements for a particular application involving publishers and subscribers, and the requirements for the pub-sub infrastructure:

- The application, comprising the publishers and subscribers. Publishers and subscribers may not trust each other, and may not trust the pub-sub network.
- The infrastructure, consisting of the pub-sub network that provides services to the application. The infrastructure may not trust publishers and subscribers. Components of the infrastructure may not necessarily trust each other.

For example, providing a mechanism that defines who has what access to what information is mostly an application-level concern. It requires a definition of identity, authorization and access control within the pub-sub infrastructure. In the meantime, controlling who is able to change the subscription database maintained by the pub-sub service and restricting channel utilization are infrastructure-level protection issues.

In this section, we explore the various security issues and requirements in the two categories. The general security needs of the application include confidentiality, integrity, and availability, while the security concerns of the infrastructure focus primarily on system integrity and

availability.

Other important issues stem from the fact that the pub-sub system is a service layer. Invariably, it must deal with applications with varying security needs and requirements. It is therefore inappropriate for the pub-sub system to dictate a global security policy that will be implemented to disseminate information for every application. A major challenge in designing a security architecture for pub-sub is the provision of a flexible security framework allowing diverse policies and mechanisms to be implemented within the same pub-sub infrastructure.

In Section 4, we discuss security issues faced by pub-sub systems that are similar to those present in traditional network security. The content-based routing and dynamic subscription properties of pub-sub systems introduce many new security requirements and challenges. While some of these issues are new, there appear to be several opportunities to adapt known solutions to address these problems. Section 5 discusses specific confidentiality concerns for pub-sub systems. Section 6 considers accountability and billing issues. Section 7 explores denial-of-service vulnerabilities particular to pub-sub systems.

4 Generic issues

Some security issues in pub-sub systems are not unlike those that appear in other distributed systems that cross administrative domains. In some cases, existing approaches can be adopted to achieve these goals, often with only minor modification. We discuss those cases below.

Authentication. Authentication establishes the identity of the originator of an action. In pub-sub, we consider two flavors of authentication, end-to-end and point-to-point. End-to-end authentication in this context means that if subscriber A receives a message claiming to have originated from publisher B, A can verify that B is indeed the publisher of the message. Point-to-point authentication is concerned only with the immediate end points of a communication: if A receives a message from B, A can verify that B is indeed the sender of the message where A and B can be publishers, subscribers or network servers.

End-to-end authentication can be implemented outside of the pub-sub domain. If a PKI exists independent of the pub-sub network, end-to-end authentication can be accomplished by having publishers sign messages using their private keys. The subscribers can then verify a publisher's identity by verifying the digital signatures attached to the message. The signing and verification operations occur outside of the pub-sub domain, and they can be administered independently.

If the pub-sub infrastructure is trusted, end-to-end authentication can be replaced with point-to-point authentication. Point-to-point authentication is a well-understood practice, and standard techniques should apply here.

There have been instances of pub-sub systems implemented using Opengroup's Distributed Computing Environment (DCE) [18] and its security features [4][21][24]. The potential size of an Internet-scale pub-sub system may give rise to scalability problems for DCE that are not present in smaller-scale systems. The pros and cons of using DCE (and other existing technologies) to outfit a pub-sub system need to be investigated closely before more informed assessment can be made.

Information integrity. The standard means to provide information integrity is by using digital signatures. A digital signature, when signed on the message digest with the sender's private key, provides two pieces of evidence: a) the message content has not been changed since it is signed, and b) the message indeed originated from the sender.

The provision of digital signatures can be largely independent of the pub-sub infrastructure.

Consider again using a PKI for publishers and subscribers. Message integrity can be enforced by having the sender digitally sign every outgoing message. The establishment and management of the PKI can be performed independently of the pub-sub layer.

Subscription integrity. In a pub-sub system, the user subscriptions kept by the network form the basis for routing and forwarding — therefore the subscriptions must be protected from unauthorized modifications. This is a traditional access-control issue that can be solved with traditional means providing proper authentication and rights management. For most realistic cases, it is reasonable to assume that subscribers can trust the pub-sub infrastructure to implement subscription functions without malice.

Service integrity. Integrity of the pub-sub service can be put at risk if malicious faults arise at the infrastructure level (e.g., infrastructure hosts are compromised). A malicious server can insert bogus subscriptions and act as a bogus subscriber to neighboring servers. Moreover, it can ignore the routing algorithm entirely and route messages to arbitrary destinations or drop them completely.

Protecting the pub-sub network from malicious intrusions is not unlike protection of other large networks. However, if the infrastructure is compromised, pub-sub systems present new research questions regarding mechanisms that preserve services in the presence of malicious infrastructure servers. This topic is investigated in depth in the following sections.

We note that this is not solely a security architecture issue. For example, one can design the routing algorithm in such a manner that there is never a single route between any pair of publisher and subscriber, and that each message is routed on multiple routes to its destination. At the price of increased resource consumption, this mechanism ensures a high probability that a message will be delivered to its intended parties despite a small number of malicious servers.

To begin tackling the problem of service integrity in the presence of malicious infrastructure hosts, a comprehensive fault analysis is needed in which the malicious faults are enumerated and consequences examined. We can then begin to understand the extent to which the existing infrastructure may be able to tolerate such malicious faults and subsequently design mechanisms to increase this tolerance.

User anonymity: User anonymity in pub-sub can be achieved with various anonymizing techniques developed for distributed systems [22][25]. It is also worth noting that the pub-sub routing and forwarding mechanism can be used as a lightweight anonymity tool. For example, in the Siena system the publications travel along a shortest path from the publisher to the subscribers [8]. Because of the way the routing mechanism works, a pub-sub server in Siena only knows its immediate predecessor and successor in the path. End-point anonymity is preserved in any path that has more than two hops. It is possible that with some strengthening, the pub-sub routing and forwarding algorithm can be used as a full-fledged anonymity tool without introducing much extra cost.

5 Confidentiality

Pub-sub systems introduce three novel confidentiality issues:

- Can the infrastructure perform content-based routing, without the publishers trusting the infrastructure with the content? (Information confidentiality)
- Can subscribers obtain dynamic, content-based data without revealing their subscription functions to the publishers or infrastructure? (Subscription confidentiality)

- Can publishers control which subscribers may receive particular publications? (Publication confidentiality)

Each of these poses new problems, but there appear to be opportunities to adapt well-known approaches towards satisfactory solutions.

Information confidentiality. When information being published contains sensitive content, publishers and subscribers may wish to keep information secret from the pub-sub infrastructure. This is especially important in a large pub-sub system where information may travel through network segments that are not necessarily trusted. Recall the resume circulation example from Section 2. It is conceivable that suppliers of resumes may wish to keep the resume content private from the routing infrastructure—an untrustworthy infrastructure server may copy every resume that routes through it and then sell them for a profit.

The requirement of confidentiality against the infrastructure is in a fundamental conflict with the pub-sub model. By definition, the pub-sub network routes information based on dynamic evaluations of information content against user subscriptions. Keeping the information private from the routing hosts may hinder such evaluations and hence routing. In particular, routing and forwarding optimizations such as the ones performed by Gryphon [3] and Siena [8] will be impossible to carry out if the infrastructure hosts do not have access to the information content.

Further note that the legitimate receivers of the information (i.e., the subscribers) must be able to read the information content, which may require an out-of-band agreement among the publishers and the subscribers so that the subscribers can recover the content of the publication (such as using a key). Incorporating an out-of-band key distribution or a similar scheme takes away the benefits of the basic pub-sub model—it follows a point-to-point communication model rather than the many-to-many model in a true publish-subscribe system.

A potentially promising technique in providing information confidentiality against the infrastructure is computing with encrypted data [1][13]. In general, a function f can be computed with encrypted data (a.k.a: f is encryptable) if there exists two functions E and D such that

- E and D are two polynomial time algorithms
- E maps x to an encrypted instance y
- D maps $f(y)$ to $f(x)$
- Nothing about x is revealed by y except what is implied by the result of $f(y)$

Abadi et al. proved that all functions in ZPP^1 are encryptable [1]. In other words, there exist E and D for every polynomial-time boolean function such that the data can be hidden from the function evaluator. However, a protocol between the publishers and subscribers is still needed so that E and D can be agreed upon and computed accordingly. We pointed out earlier that this conflicts with the many-to-many communication model. In addition, these secure computation protocols are often computationally intensive and require a large amount of communication overhead, which could be prohibitively expensive to carry out.

Subscription confidentiality. User subscriptions can reveal sensitive information about the user, in which case the subscriber may wish to keep the subscriptions private. Consider the human resource resume example. An HR person, upon being told that her company is starting a top-secret new project, wants to enter a new subscription that allows her to receive resumes with a particular background. Because of the sensitive nature of the project, she may wish to keep her

¹ The class ZPP consists of boolean functions that can be computed in polynomial time with zero probability of error.

subscription private even from the pub-sub system—after all, the system may turn around and sell this knowledge to her competitors.

More formally, subscription confidentiality against the infrastructure can be viewed as follows:

The subscriber S would like the network N to compute $f(x)$ without revealing f to N . Here, x is the publication information and f is the subscription function.

A closely related topic of interest to subscription confidentiality is secure circuit evaluation [2], which has been studied in various models [4][9][27]. Secure circuit evaluation hides the circuit² from the circuit evaluator. In theory, if computing with encrypted data can be achieved, hiding the circuit can be implemented as encoding the circuit itself as an input to a universal circuit evaluation function [2]. In practice, however, it is difficult and often impractical to encode the function as an input to a universal circuit; the proposed schemes often involve an expensive protocol.

Another related subject of interest is Private Information Retrieval (PIR) [11][12]. PIR mechanisms allow a user to retrieve records from a database, and in the meantime, hide what she retrieves from the database. Studies on PIR schemes showed that PIR is at least as hard as Oblivious Transfer [12], which implies the existence of one-way functions. A close examination of subscription confidentiality suggests that close relations exist between PIR and subscription confidentiality. For example, one can easily construct a PIR mechanism using a black box that implements subscription confidentiality simply by inputting every database record as a publication into the black box. Conversely, a simple case of a subscription function that matches a finite number of pre-defined strings can be reduced to PIR with publications modeled as databases and subscriptions as PIR queries.

The construction of PIR from subscription confidentiality suggests that the latter cannot be achieved using weak computational primitives—it is at least as hard as PIR schemes. A more general reduction from subscription confidentiality to PIR can be the starting point of constructing realistic confidentiality mechanisms to hide user subscriptions from the pub-sub infrastructure. However, all PIR-based schemes move some filtering operations from the database to the user, which implies more communication overhead as well as user-side computation load. Therefore challenges regarding performance and efficiency will still remain even if a general reduction can be constructed.

It is worth noting that the combination of information and subscription confidentiality against the infrastructure achieves a fairly strong level of user privacy—the most other people can deduce is that you are associated with this particular pub-sub system, however, they will not be able to find out how you are using the service (e.g., publishing what or subscribing to what). This can be a viable alternative to straightforward user anonymity.

Publication confidentiality. In many pub-sub applications, publishers do not know and perhaps do not care to know the identity of the subscribers who receive their information. In those applications, there is no need for subscription control — anybody can subscribe to anything. In other applications, however, it is important that publications be kept secret from ones who are not legitimate subscribers (the concept of legitimacy should be application specific). Consider the stock quote example. For billing purposes, quotes should be sent to paying customers only. Therefore they must be kept confidential from other users.

Publication confidentiality can be handled independent of the pub-sub infrastructure. For

² Circuit here means a function that can be represented as a binary circuit.

example, the publisher can distribute a group key to the subscribers using some out-of-band channel and encrypt the information content with the key. This ensures that only the subscribers with the right key can read the message. The drawback of this scheme is obvious: setting up a group key a priori, in essence, transforms the communication model into a traditional multicast model, and therefore minimizes the benefits of publish and subscribe.

Alternatively, publishers can trust the infrastructure to maintain publication confidentiality. For example, instead of registering with the stock quote provider, a user can register with the pub-sub system for the stock quote service. Publishers enter the quotes into the system without knowing who will be receiving them. It is then up to the pub-sub system to ensure that only registered users receive the appropriate quotes.

A potential solution here is to let the application choose an appropriate mechanism. That is, applications that do not care about publication confidentiality should not have to pay the cost associated with exerting confidentiality control. Meanwhile, for applications that desire publication confidentiality, the pub-sub layer must provide a) an interface for the application to specify a control policy, and b) a mechanism that supports such policies. Designing such a flexible security framework is no trivial undertaking—the interface must be expressive and easy to use, and the system must be prepared to carry out a whole spectrum of mechanisms desired by the applications. There is also the question of whether to implement an inexpensive policy as the default case—the default case can always be overwritten, but the specifics of a default policy must be carefully laid out so that it does not detract from the system flexibility.

6 Accountability

In commercial pub-sub applications, publishers may want to charge subscribers for the information they provide. The nature of the pub-sub system, however, means there may be no direct relationship between a publisher and subscriber. Further, a publisher has no way of knowing which subscribers receive (and should be charged for) particular datagrams.

Out-of-band solutions. The most obvious solution to accountability is for publishers to bill subscribers by selling keys that decrypt selected data, which is similar to the publication confidentiality discussed earlier. Again consider the stock quote example. A subscriber could pay the supplier of stock quotes a monthly fee for the relevant keys. Publishers would send quotes into the pub-sub network encrypted with the appropriate key so that only subscribers who had paid for that information would be able to decrypt it. This ensures only paying subscribers will be able to view the information, but sacrifices many of the advantages of a pub-sub network. It requires subscribers to reveal their identity and interests to publishers, and demands a direct publisher-subscriber relationship. Further, it eliminates the possibility of per-data payment schemes and dynamic subscriptions.

Infrastructure-based accountability. If both subscribers and publishers trust the pub-sub infrastructure to account fairly, the pub-sub infrastructure can bill subscribers according to the amount of information they receive and pay publishers according to the information they provide without there being any direct relationship between publishers and subscribers. In the stock quote application, the infrastructure would keep track of who received what quotes at what frequency and who publishes them. Periodically the system would bill the subscribers and send a portion of the payment to the appropriate publishers.

This type of account management can be carried out at the entry points of the system where the network interacts with the publishers and the subscribers. An important issue here is for the

system to demonstrate that its accounting is conducted in a fair way (see the discussion of auditability below.)

In addition to user account management, accountability can also extend to pub-sub servers when the message routing network spans distinct domains that include possibly competing and mutually suspecting organizations. In these cases, auditing needs to take place at the interfacing points of the various subsystems.

The various issues with accountability in pub-sub remain much the same as those in other distributed systems. The techniques in these other systems, such as the market-based pricing account management scheme in [20], can be adopted here (see [6][20]).

Auditability. If the pub-sub infrastructure is used to bill subscribers and reward publishers, publishers may wish to audit the accounting to verify that they are being paid fairly for the subscriptions they satisfy. Complete auditability is impossible if we wish to provide subscription confidentiality. Auditing based on statistical sampling at trusted points within the pub-sub network could provide a satisfactory solution, however. A publisher could examine logs from points in the network and compare the number of datagrams present to the fees collected from the pub-sub infrastructure. This would give the publisher a confident lower bound on the number of subscriptions satisfied. The pub-sub infrastructure could still cheat when a single datagram satisfies many subscriptions. However, if the log points were located throughout the network, it would be difficult for the pub-sub infrastructure to conduct any large-scale cheating without being detected. There is a natural tension between auditability (which improves as log points are moved closer to subscribers) and subscriber confidentiality (which is compromised by logging close to subscribers).

Another potential scheme for auditing is the use of verifiable secure computation [15]. If the accounting operation is viewed as a function and the logs are an input to the function, verifiable secure computation techniques allow the result of the accounting to be verified without disclosing the inputs (logs) to the function.

7 Availability

As in other communication systems, denial-of-service attacks remain as a significant risk for pub-sub systems. In addition to the standard infrastructure attacks to which all distributed applications are vulnerable (as discussed in Section 4), pub-sub systems open up some new classes of attacks. In particular, malicious publications and subscriptions can be used to overload the system.

Denial-of-service attacks are impossible to prevent in the general case. However, certain measures can be taken to minimize the probability of a wide spread denial-of-service attack. We discuss three different measures here. From the simplest to the most sophisticated, each scheme results in a certain level of publication bandwidth control.

Limited publication. This is a straightforward measure that simply limits the size of each publication [25]. Variations of this scheme may limit the frequency of publication or both size and frequency.

CPU-cycle-based payment scheme. This is a more sophisticated scheme that combines payment with publication control. Hash Cash is one such scheme [17]. The basic idea behind Hash Cash and other CPU cycle based payment schemes is that it requires the publisher to perform some complex computational tasks (e.g. finding collisions in the hash function) before publishing. The more complex the task, the more control the system has over the publication

process.

Customized publication control. The primary drawback of the previous two schemes is that they are both one-size-fits-all solutions and therefore lack the flexibility of differentiating among different publications. A customized publication control does just that; it allows subscribers to specify which publisher is allowed to publish information. Bogus publications can be weeded out at the system entry points. Consider the scenario in which a subscriber wishes to accept notifications from publishers who know a certain secret. This criterion can be expressed as part of the subscription. In addition to the normal subscription function, the subscriber specifies a challenge clause, which the servers use to challenge the sender of the publications. More specifically, consider two functions, f and g , with the following two properties:

- f and g are hard to invert,
- $g(f(x), f(y)) = f(g(x,y))$

The publish and subscribe protocol behaves as follows: When subscriber A initiates a challenged subscription, A establishes a filter that includes g as the challenge function.

- preprocessing step: All legitimate publishers know a secret function f , which is distributed in an out-of-band method.
- subscriber \rightarrow network: {subscription, g }
- publisher \rightarrow network: $\{x, f(x)\}$, x is a random number chosen by the publisher.
- network \rightarrow publisher: $\{y, g(x, y)\}$, y is a random number chosen by the network and g is the challenge function in the subscription.
- publisher \rightarrow network: $\{f(y), f(g(x,y))\}$
- the network server can now compute $g(f(x), f(y))$ and compare that with $f(g(x,y))$. The server allows publication only if the two match.

In the last step, the network server verifies that the publication indeed should be forwarded to the subscribers by engaging in a challenge-and-response protocol with the publisher. With this scheme, bogus notifications will not generate unnecessary message traffic within the network. Note that this scheme is very similar to a public-key authentication mechanism. The only difference is that we might be able to find two functions f and g that are more efficient than the public key operations. A more detailed description of such an authentication mechanism can be found in [26].

We note that subscribers can specify any arbitrary function in place of the challenge-and-response example. Also note that such a function can be easily incorporated as an extension to the subscription semantics—basically as an extension to the language to allow the specification of an extra clause. The more complex the function, the more expressive the subscription language needs to be, which will further constrain the types of optimization the network is able to perform in terms of routing and forwarding. The tradeoff between the power of publication control versus amenability for optimization must be examined if such mechanism is to be adopted.

8 Summary

This paper presents a general discussion of the security issues and requirements in an Internet-scale pub-sub system. The dynamic content-based routing mechanism in such networks poses

both opportunities and challenges for security. The nature of pub-sub systems present several apparent security paradoxes, among them routing based on content while keeping the content confidential, routing based on subscriptions without revealing the subscription functions, accounting based on subscriptions satisfied without revealing the subscription functions. We have not presented full solutions to any of these problems, but have suggested approaches based on new applications of well-known mechanisms that may lead to solutions.

We intend this paper to be an initial roadmap of establishing a comprehensive security architecture for large-scale pub-sub systems. These systems raise numerous interesting and important security issues for further research.

9 Acknowledgements

The authors thank John Knight for discussions that helped shaping some of the ideas here.

The work of Chenxi Wang was completed when she was a research scientist at the University of Virginia. Her work was supported in part by the Defense Advanced Research Agency, under the agreement number F30602-96-1-0314.

The work of David Evans was supported by in part by the National Science Foundation and NASA under agreement numbers NSF CCR-0092945 and NASA NRC 99-LaRC-4.

The work of A. Carzaniga and A.L. Wolf was supported in part by the Defense Advanced Research Projects Agency, Air Force Research Laboratory, Space and Naval Warfare System Center, and Army Research Office under agreement numbers F30602-01-1-0503, F30602-00-2-0608, N66001-00-1-8945, and DAAD19-01-1-0484.

The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory, Space and Naval Warfare System Center, Army Research Office, National Science Foundation, NASA or the U.S. Government.

10 References

- [1]. M. Abadi, J. Feigenbaum, and J. Kilian. "On Hiding Information from an Oracle". In the proceedings of the Ninth Annual ACM Conference on Theory of Computing. May, 1987. New York, pages 195-203.
- [2]. M. Abadi and J. Feigenbaum. "Secure Circuit Evaluation". Journal of Cryptology, Vol 2. No. 1: pages 1-12. 1990.
- [3]. M. Aguilera, R. Strom, D. Sturman, M. Astley, and T. Chandra. "Matching Events in a Content-based Subscription System". In the conference proceedings of the Principles of Distributed Computing, 1999.
- [4]. M. Ben-Or, S. Goldwasser, and A. Wigderson. "Completeness Theorems for Non-cryptographic Fault-tolerant Distributed Computation," Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pages 1-10.
- [5]. B. Blakley. "CORBA security". Addison-wesley. August 1999.

- [6]. S. Brands. “Untraceable off-line cash in wallets with observers”, In Advances in Cryptology: Proceedings of Crypto’93, pages 302-318. Springer-Verlag (LNCS773), 1993.
- [7]. A. Carzaniga, D. Rosenblum and A. Wolf. “Achieving Scalability and Expressiveness in an Internet-Scale Event notification service”. In Proceedings of the 2000 ACM Conference of PODC 2000. Portland Oregon. Pages 219 –227. 2000.
- [8]. A. Carzaniga, D. Rosenblum and A. Wolf. “Design and Evaluation of a Wide-Area Event Notification Service”. ACM Transactions on Computer Systems, 19(3):332-383, Aug 2001.
- [9]. S. Chapin, C. Wang, W. Wulf, and A. Grimshaw. “A New Security Model for Distributed Meta Systems”. In Future Generations of Computer Science. October 1998.
- [10]. D. Chaum, C. Crepeau, and I. Damgard. “Multiparty Unconditionally Secure Protocols,” In the proceedings of the 20th Annual ACM Symposium on Theory of Computing, 1988, pages 11-19.
- [11]. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. “Private Information Retrieval”. In the proceedings of the 36th IEEE Conference on the Foundations of Computer Science (FOCS). Pages 41-50. October 1995.
- [12]. G. Di Crescenzo, T. Malkin, and R. Ostrovsky. “Single-database private information retrieval implies oblivious transfer”. In Advances in Cryptology—EUROCRYPT’00.
- [13]. J. Feigenbaum. “Encrypting problem instances, or..., Can you take advantage of someone without having to trust him?” In the proceedings of Crypto’ 85, Springer-Verlag, 1986, pages 477-488.
- [14]. M. Franklin and M. Yung. “Secure and Efficient Off-line Digital Money”. In the proceedings of the Annual International Colloquium on Automata, Languages and Programming. 1993.
- [15]. R. Gennaro, S. Micali, “Verifiable Secret Sharing as Secure Computation”, Eurocrypt '95, LNCS 921, Springer-Verlag, Berlin 1995, pages 168-182.
- [16]. Gnutella. <http://www.gnutellanews.com>.
- [17]. Hash Cash. <http://www.cypherspace.org/~adam/hashcash>.
- [18]. Opengroup. Distributed Computing Environment. <http://www.opengroup.org/dce>.
- [19]. D. Rosenblum and A. Wolf. A design framework for Internet-scale event observation and notification. In Proceedings of the Sixth European Software Engineering Conference. LNCS, number 1301, pages 344-360. Springer-Verlag, 1997.
- [20]. J. Sairamesh, D. Ferguson, and Y. Yemini, “An Approach to Pricing, Optimal Allocation, and Quality of Service Provisioning in High-Speed Networks”, In the Proceedings of INFOCOM’95.
- [21]. Bill Segall and David Arnold. “Elvin has left the building: A publish/subscribe notification service with quenching”. In the Proceedings of

- AUUG '97, Brisbane, Australia, September 1997.
- [22]. P.F. Syverson, D.M. Goldschlag, and M.G. Reed. "Anonymous connections and onion routing". In Proceedings of the 1997 IEEE Symposium on Security and Privacy, May 1997.
 - [23]. Transarc, IBM white paper. June 1998. <http://www.transarc.ibm.com/library>
 - [24]. Talarian middleware whitepaper. <http://www.talarian.com>
 - [25]. M. Waldman, A. Rubin, L. Cranor. "Publius, A robust, tamper-evident, censorship-resistant web publishing system". In the proceedings of the 9th USENIX Security Symposium. August, 2000.
 - [26]. W. Wulf, A. Yasinsac, K. Oliver, and R. Peri. "Remote Authentication without prior shared knowledge". In the proceedings of the Network and Distributed Systems Security Conference, February 1994. San Diego, California.
 - [27]. A. Yao. "How to generate and exchange secrets," Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, 1986, pages 162-167.